

# Generierung von Java Klassen aus einer XML Schemabeschreibung (XSD)

Generative Software Development – Projektvorstellung

Andreas Linz



Universität Leipzig

3. Juni 2014

## 1 Einführung

- Idee
- Motivation
- XSD

## 2 Codegenerator

- Datenmodelle
- Template-Engine

## 3 Schlussbetrachtung

- Alternative zur Template-Engine
- Generatorsystem
- Fazit

- Codegenerator der aus einer abstrakten Datenformatbeschreibung ein Java Package erzeugt<sup>1</sup>
- Datenformatbeschreibung als *XML Schema Description*
- Ändern der Zielsprache sollte mit vertretbarem Aufwand möglich sein

---

<sup>1</sup>Implementierung existiert bereits als Teil von JAXB

## Probleme

- Daten müssen für die Kommunikation mit Webservices serialisiert werden
- in der Regel wird XML unterstützt
- Struktur der erwarteten Daten kann als *XML Schema Description* maschinenlesbar definiert werden
- *Mapping* zwischen XML- und Klassendarstellung in der gewählten OO-Sprache um Daten einfach verarbeiten zu können

## Generatorlösung

- Generierung der Klassendarstellung aus XSD in der gewünschten OO-Sprache (in diesem Fall Java)
- Erzeugen von Serialisierungs- und Deserialisierungsmethoden<sup>2</sup>

---

<sup>2</sup>Werden derzeit nicht generiert.

# XML Schema Description – XSD

- XML Schema Description (kurz XSD aber auch XML Schema) ist eine Schemabeschreibungssprache welche Regeln enthält um den Aufbau von XML-Daten zu definieren
- Schemabeschreibung wird üblicherweise genutzt um XML-Daten gegen ein Schema zu validieren
- XSDs sind selbst gültiges XML
- Es können Datentypen für XML-Element und Elementattribute definiert werden

Listing 1: Minimalbeispiel für ein XML-Element

```
1 <element attribut="wert">Inhalt</element>
```

Listing 2: Beispiel für einen einfachen Schematyp aus [Fac14]

```
1 <xsd:element name="auth_createToken_response"  
2   type="auth_token" />  
3  
4 <xsd:simpleType name="auth_token">  
5   <xsd:restriction base="xsd:string" />  
6 </xsd:simpleType>  
7  
8 <!-- Beispiel für eine Instanz des Typs -->  
9 <auth_createToken_response>foobar</auth_createToken_response>
```

## Xsd strukturierte Typen

Listing 3: Beispiel für einen strukturierten Schematyp aus [Fac14]

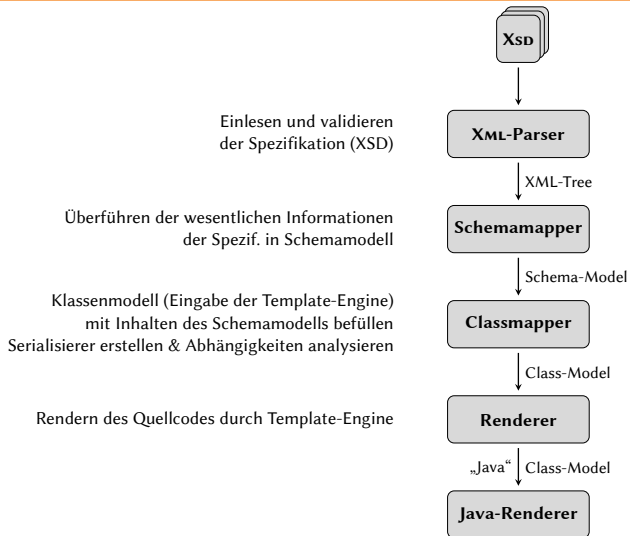
```
1 <xsd:element name="video_getUploadLimits_response"  
2   type="video_limits" />  
3  
4 <xsd:complexType name="video_limits">  
5   <xsd:sequence>  
6     <xsd:element name="length" type="xsd:int" />  
7     <xsd:element name="size" type="xsd:long" />  
8   </xsd:sequence>  
9 </xsd:complexType>  
10  
11 <!-- Beispiel für eine Instanz des Typs -->  
12 <video_getUploadLimits_response>  
13   <length>21</length>  
14   <size>42</size>  
15 </video_getUploadLimits_response>
```



- Generator<sup>3</sup> ist in Python 3.4 implementiert
- Zielsprache ist Java
- Überführung der XML Schemabeschreibung in internes Datenmodell welches einer Template-Engine als Eingabe dient
- Änderung der Zielsprache über Anpassung der Templates

---

<sup>3</sup>Codegenerator u. Generator wird hier Synonym verwendet



## Schemamodell

- kapselt die Spezifikation, in diesem Fall die Regeln aus der XSD-Datei
- XML-Parser<sup>4</sup> erzeugt Objektbaum und validiert Xsd
- Über Objektbaum iterieren und wesentliche Informationen in Schemamodell übernehmen (XML spezifische Inhalte weglassen)
- Schemamodell enthält Attribut-, Element- und Typdefinitionen sowie Namensraumangaben

---

<sup>4</sup>lxml

## Klassenmodell

- Bildet Definitionen und Regeln des Schemamodells auf Konstrukte der Zielsprache ab
- Logik in Templates vermeiden, verringern die Wartbarkeit
- enthält Abhängigkeiten zwischen den Definitionen (Importieren von Klassen siehe listing 6)
- enthält Serialisierungsmethoden<sup>5</sup>

---

<sup>5</sup>Deserialisierer aus Zeitgründen nicht mit generiert

# Template-Engine

- Eine Template-Engine ist ein Textersetzungs-system welches „Templates“ (Vorlagen) verarbeitet und darin enthaltene Platzhalter durch andere Inhalte ersetzt
- Vom Generator wird die **Mako** Template-Engine verwendet
- Einhalten von Namenskonventionen (camelCase) durch Formatierungsmethoden
- Ordnerstruktur des generierten Java Packages wird aus der Namensraumangabe generiert, bspw. `http://api.facebook.com/1.0/` wird zu:  
`%ausgabepfad%/api.facebook.com/1_0/...`  
`...src/main/java/com/facebook/api/`

## Listing 4: Beispieltemplate

```
1 public class ${classname} {  
2     ${field.modifier} ${field.type} ${field.name}  
3     ${'=_'+field.value if field.value else ''};  
4     ...  
5 }
```

# Beispieldefinition für generierte Klasse

Listing 5: Beispieldefinition für generierte Klasse von Listing 6 (aus [Fac14])

```
1 <xsd:complexType name="video_tag">
2   <xsd:sequence>
3     <xsd:element name="vid" type="vid" />
4     <xsd:element name="subject" type="uid" />
5     <xsd:element name="created_time" type="time" />
6     <xsd:element name="updated_time" type="time" />
7   </xsd:sequence>
8 </xsd:complexType>
```

# Beispiel für generierte Klasse

Listing 6: Beispiel für eine generierte Java-Datei (gekürzt)

```
1 package com.facebook.api;
2
3 import com.facebook.api.Uid;
4 import com.facebook.api.Vid;
5 import com.facebook.api.Time;
6
7 class VideoTag {
8     private Vid vid;
9     private Uid subject;
10
11     ...
12     public void setVid(Vid vid) {
13         this.vid = vid;
14     }
15     public Vid getVid() {
16         return this.vid;
17     }
18     ...
19     public Time getUpdatedTime() {
20         return this.updated_time;
21     }
22     public String toXML() {
23         return this.vid.toXML() + this.subject.toXML()
24             + this.created_time.toXML() + this.updated_time.toXML();
25     }
26 }
```

# Erweiterungs- und Verbesserungsmöglichkeiten

## Alternative zur Template-Engine

- Implementierung eines Sprachenmodells welches die Konstrukte der Zielsprache abbilden kann (Expressions, Statements, Conditions, Loops, ...)
- Sprachenmodell würde zu erzeugenden Code als *Abstract Syntax Tree* enthalten
- Sprachenmodell würde Semantik und ein Rendermodul die Syntax kapseln
- Vorteile:
  - Erweiterung um zusätzliche Zielsprachen durch Implementierung weiterer Rendermodules
  - Formatierung des erzeugten Codes über Parameter änderbar (Einrückungstiefe, Klammerpositionen, ...)
  - Optimierung des zu erzeugenden Codes durch Analyse des AST



## WADL

- Generieren einer gesamten Client-Bibliothek für einen RESTful Webservice
- Nutzung einer kompletten maschinenlesbare Beschreibung des Webservice als WADL (Web Application Description Language) in Verbindung mit XSD
- Generatorsystem bestehend aus:
  - Codegenerator der Bibliothek für Zugriff auf Webservicesressourcen generiert
  - Generator zur Erstellung der Datenklassen aus XSD<sup>6</sup>

---

<sup>6</sup>analog dem hier vorgestellten Generator

- bildet derzeit nicht alle in XSD erlaubten Regeln ab (sehr komplexe Typdefinitionen möglich, Anonyme Typen, ...)
- mit überschaubaren Änderungen produktiv einsetzbar
- Vertiefung der Kenntnisse im Umgang mit Template-Engines und XML-Bibliotheken in Python3

**Fragen?**