Generierung von Java Klassen aus einer XML Schemabeschreibung (XSD)

Generative Software Development – Projektvorstellung

Andreas Linz

Universität Leipzig

3. Juni 2014

Inhaltsverzeichnis

Idee

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

[1] Implementierung existiert bereits als Teil von JAXB

Motivation

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

Motivation

Generatorlösung

- Generierung der Klassendarstellung aus XSD in der gewünschten OO-Sprache (in diesem Fall Java)
- Erzeugen von Serialisierung- und Deserialisierungsmethoden[2]

[2] 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>
```

Xsd einfache Typen

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

```
<xsd:element name="auth_createToken_response"
    type="auth_token" />

<xsd:simpleType name="auth_token">
    <xsd:restriction base="xsd:string" />
</xsd:simpleType>

<!-- Beispiel für eine Instanz des Typs -->
<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>
```

Codegenerator

- Generator[3] 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

[3] Codegenerator u. Generator wird hier Synonym verwendet

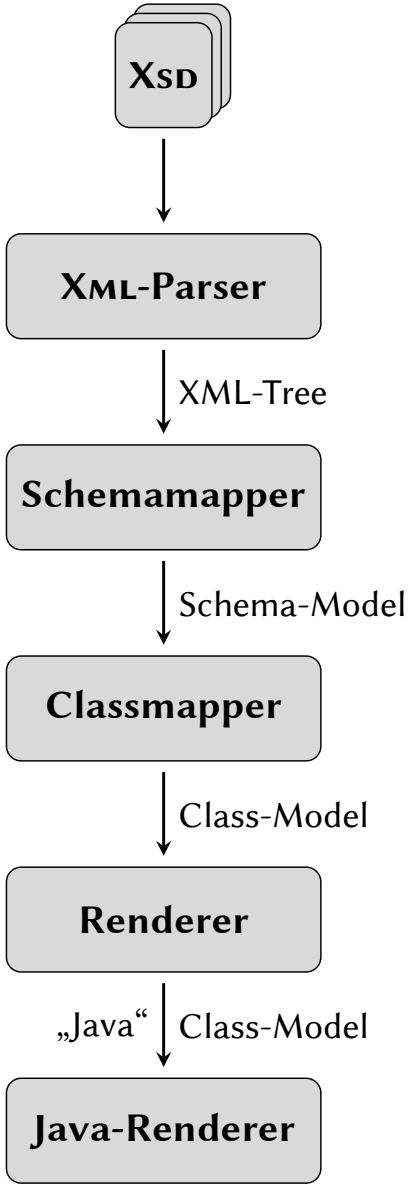
Generatorablauf

Einlesen und validieren der Spezifikation (XSD)

Überführen der wesentlichen Informationen der Spezif. in Schemamodell

Klassenmodell (Eingabe der Template-Engine) mit Inhalten des Schemamodells befüllen Serialisierer erstellen & Abhängigkeiten analysieren

Rendern des Quellcodes durch Template-Engine



Datenmodelle

Schemamodell

- kapselt die Spezifikation, in diesem Fall die Regeln aus der Xsd-Datei
- XML-Parser[4] erzeugt Objektbaum und validiert Xsd
- Über Objektbaum iterierern und wesentliche Informationen in Schemamodell übernehmen (Xml spezifische Inhalte weglassen)
- Schemamodell enthält Attribut-, Element- und Typdefinitionen sowie Namensraumangaben

[4] lxml

Datenmodelle

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[5]

[5] Deserialisierer aus Zeitgründen nicht mit generiert

Template-Engine

- Eine Template-Engine ist ein Textersetzungssystem 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     public void setVid(Vid vid) {
12         this.vid = vid;
13     }
14     public Vid getVid() {
15         return this.vid;
16     }
17 ...
18     public Time getUpdatedTime() {
19         return this.updated_time;
20     }
21     public String toXML() {
22         return this.vid.toXML() + this.subject.toXML()
23             + this.created_time.toXML() + this.updated_time.toXML();
24     }
25 }
```

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 Webserviceressourcen generiert
  - Generator zur Erstellung der Datenklassen aus XSD[6]

[6] analog dem hier vorgestellten Generator

Fazit

- 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?