Generative Software Development

Generierung von Java Klassen aus einer XML Schemabeschreibung (XSD)

Andreas Linz

Softwaresystemfamilien — SS 2014
Universität Leipzig

31. Mai 2014

Inhaltsverzeichnis

1 Einführung

Dieses Dokument beschreibt den im Rahmen der Vorlesung *Softwaresystemfamilien* erstellten Codegenerator, welcher aus einer abstrakten Beschreibung von Datenformaten in Form einer XML Schemabeschreibung (nachfolgend XSD) ein Java Package erzeugt.

1.1 Motivation

Um Daten mit Webservices austauschen zu können, müssen diese für den Übertragungsweg serialisiert werden. In der Regel wird dabei XML als Format für die serialisierten Daten unterstützt. Die Struktur der serialisierten Daten, die von dem Dienst empfangen bzw. an diesen gesendet werden können, kann in Form einer XML Schema Description (kurz XSD) maschinenlesbar definiert werden.

Für eine effektive Verarbeitung der Daten in einer objektorientierten Programmiersprache (bspw. Java), benötigt man eine Abbildung aus der Beschreibung der XML-Darstellung in eine Darstellung als Klassen der gewählten Programmiersprache. Die manuelle Erstellung dieses *Mappings* ist einerseits monoton und deshalb fehlerträchtig, sowie andererseits sehr zeitaufwendig, sollte sich die API und die verwendeten Datenstrukturen des Webservice ändern. Dies hätte zur Folge das dieses Mapping wieder manuell angepasst werden müsste.

Der Generator übernimmt diese Arbeit und erzeugt aus der Beschreibung des Datenschemas automatisch die Klassendarstellung in der gewünschten Programmiersprache und erstellt zusätzlich noch eine Serialisierungsmethode um die Klassendarstellung wieder zurück nach XML wandeln zu können[1].

Wirklich spannend wie dieses Konzept erst als Teilaspekt eines *Generatorsystems*, welches eine komplette Client-Bibliothek für einen RESTful Webservice erzeugt, falls dieser neben der abstrakten Beschreibung der erwarteten Datenformate auch eine Beschreibung der verfügbaren Ressourcen bereitstellt, bspw. als WADL-Datei (siehe Abschnitt 4.1 S. 8).

1.2 XML Schema Description (XSD)

Die *XML Schema Description* o.a. nur *XML Schema* ist eine Schemabeschreibungssprache welche Regeln enthält um den Aufbau einer XML-Datei zu definieren. Üblicherweise wird diese Schemabeschreibung dazu genutzt um XML-Daten gegen dieses Schema zu validieren. Eine XML Schemabeschreibungsdatei ist selbst gültiges XML.

XSD erlaubt die Typdefinition für XML-Elemente und Elementattribute als einfache (sogenannte `simpleTypes`) sowie strukture Typen (`complexTypes`).

Listing 1: Minimalbeispiel für ein XML-Element

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

[1] Um wirklich effektiv arbeiten zu können wäre auch noch die Generierung von Deserialisierungsmethoden wünschenswert, um eine Instanz der Klasse direkt aus der XML-Form zu initialisieren. Leider hätte dies aber nicht mehr innerhalb der Abgabefrist implementiert werden können.

Listing 2: Beispiel für einen einfachen Schematyp [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>
```

Listing 3: Beispiel für einen strukturierten Schematyp [Fac14]

```
<xsd:element name="video_getUploadLimits_response" type="video_limits" />

<xsd:complexType name="video_limits">
    <xsd:sequence>
        <xsd:element name="length" type="xsd:int" />
        <xsd:element name="size" type="xsd:long" />
    </xsd:sequence>
</xsd:complexType>

<!— Beispiel für eine Instanz des Typs —>
<video_getUploadLimits_response>
    <length>21</length>
    <size>42</size>
</video_getUploadLimits_response>
```

Neben Xsd existieren noch weitere Schemasprachen wie *RelaxNG*, die aber im Rahmen dieser Arbeit nicht behandelt werden.

2 Codegenerator

> *Ein Codegenerator ist ein Programm, welches aus einer höhersprachigen Spezifikation[2], einer Software oder eines Teilaspektes die Implementierung erzeugt.* *(nach [CE00, S. 333])*

Im Allgemeinen wird der Begriff „Generator“ für verschiedene Technologien verwendet, u.a. Compiler u. Präprozessoren, Template-Metaprogramming in C++ und natürlich Codegeneratoren.

Der hier beschriebene Generator erzeugt ausführbaren Code, durch die Überführung einer XML Schemabeschreibung in ein internes Datenmodell welches als Eingabe für eine Template-Engine dient (siehe Abschnitt 2.2 auf 6).

Der Codegenerierungsprozess erfolgt dabei durch die folgenden Schritte:

1. Einlesen und validieren der Spezifikation (Xsd)

[2] mit anderen Worten: auf einem höheren Abstraktionslevel als das der zur Implementierung verwendetenProgrammiersprache

2. Überführen der wesentlichen Informationen der Spezifikation in ein Schemamodell
3. Befüllen des Eingabemodells des Generators (hier als Klassenmodell bezeichnet) mit den Spezifikationen aus dem Schemamodell
4. Analysieren von Abhängigkeiten innerhalb des Klassenmodells
5. Rendern des Quellcodes durch die Template-Engine

2.1 Datenmodell

Das Datenmodell beinhaltet die Informationen der Spezifikation und bildet somit die Basis für den Codegenerator. Vom Generator werden zwei Datenmodelle zur Erzeugung des Quellcodes verwendet, dass Schemamodell (Abschnitt 2.1.1) und das Klassenmodell (Abschnitt 2.1.2).

2.1.1 Schemamodell

Das Schemamodell kapselt die in der Spezifikation enthaltenen Informationen, in diesem Fall die Regeln aus der XSD-Datei. Im ersten Schritt wird durch einen XML-Parser aus dem XSD ein Objektbaum erzeugt. Gleichzeitig wird durch den Parser (lxml [Beh14]) geprüft ob die Spezifikation selbst wohlgeformtes XML ist. Der vom Parser erzeugte Baum enthält noch viele XML spezifische Informationen welche im folgenden *mapping* Schritt wegfallen. Das `schemamapper` Modul des Codegenerators iteriert hierfür über den Objektbaum und extrahiert die relevanten Informationen, wie bspw. Element- und Typdefinitionen welche darauffolgend in das Schemamodell eingefügt werden. Das Schemamodell selbst ist ein Python *Dictionary*[3] welches Angaben über die definierten Namensräume sowie die Attribut-, Element- u. Typdefinitionen enthält.

2.1.2 Klassenmodell

Um möglichst wenig Logik innerhalb der *Templates* (siehe Abschnitt 2.2 S. 6) zu haben wird durch das Modul `classmapper` aus dem Schemamodell das Klassenmodell erzeugt. Dieses bildet die Definitionen und Regeln des Schemamodells auf Konstrukte der Zielsprache ab. Hierbei muss erwähnt werden das die XML Schemabeschreibung sehr komplexe Regeln zur Einschränkung von Wertebereichen und Definition von Typen erlaubt. Darunter fallen anonyme Typdefinitionen sowie Auftrittsreihenfolgen u. -häufigkeiten für Elemente [W3C04]. Dieses Regeln sind im Schemamodell enthalten, werden aber im Klassenmodell nicht übernommen da deren Abbildung den zeitlichen Rahmen diese Projektes übersteigen würde.

[3] auch als *assoziatives Array* o. *Map* bezeichnet

Durch das `classmapper` Modul werden die Element- und Typdefinitionen auf Abhängigkeiten untersucht und diese als `dependencies` Dictionary dem jeweiligen Klassenmodell hinzugefügt. Da aus jeder Element- und Typdefinition eine eigene Java-Klasse generiert wird müssen darin verwendete Typen, die in einer anderen Klasse-/Datei definiert sind, importiert werden. Diese Importanweisungen werden aus den `dependencies` generiert (siehe Zeile 3-5 in Listing 5, S. 6). Außerdem werden referenzierte Typen aufgelöst und durch deren Definition ersetzt. Zusätzlich werden vom `classmapper` auch Serialiserungsmethoden erstellt, die aus der erzeugten Java-Klasse wieder XML generieren.

2.2 Template-Engine

Eine Template-Engine ist ein Textersetzungssystem was Templates (Vorlagen) verarbeitet und darin enthaltene Platzhalter durch andere Inhalte ersetzt. Das `javaRenderer` Modul dieses Codegenerators verwendet dabei die *Mako Template-Engine* [mak14]. Als Eingabe dient dabei das Klassenmodell mit welchem die Templatedateien für Element- bzw. Typdefinitionen befüllt werden. Listing 5 zeigt Java-Klasse welche aus der Typdefinition von Listing 4 generiert wurde. Das `javaRenderer` Modul übergibt der Template-Engine Methoden zur Formatierung von Klassen- und Methodennamen um bestimmte Namenskonventionen einzuhalten. In diesem Fall Methoden zur Erzeugung von *camel-cased* Namen für Klassen und Methodennamen. Dies ist für die Funktionsweise nicht notwendig, unterstützt aber die Lesbarkeit und Akzeptanz des erzeugten Codes. Der Name des erzeugten Java Packages wird ebenso wie die Ordnerstruktur[4] des Projektes aus der Angabe des Zielnamensraums (targetnamespace) in der Schemadefinition erzeugt. Aus dem Zielnamensraum `http://api.facebook.com/1.0/` würde folgende Ornderstruktur generiert werden:
`%ausgabepfad%/.api.facebook.com/1_0/src/main/java/com/facebook/api/`.

Listing 4: Beispiel für eine XSD Typdefinition 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>
```

Listing 5: Beispiel für eine generierte Java-Datei (getter und setter 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;
```

[4] Nach Maven - Standart Directory Layout

```
6
7  class VideoTag  {
8
9      private Vid vid;
10     private Uid subject;
11     private Time created_time;
12     private Time updated_time;
13
14
15     public void setVid(Vid vid) {
16         this.vid = vid;
17     }
18
19     public Vid getVid() {
20         return this.vid;
21     }
22     ...
23     public Time getUpdatedTime() {
24         return this.updated_time;
25     }
26
27     public String toXML() {
28         return this.vid.toXML() + this.subject.toXML()
29             + this.created_time.toXML()
30             + this.updated_time.toXML();
31     }
32 }
```

2.3 Alternative zum templatebasierten Ansatz

Alternativ zur Verwendung einer Template-Engine ist die Erstellung eines Modells für die Zielsprache des zu erzeugenden Codes möglich. Dieses *Sprachenmodell* müsste dabei die Konstrukte der Zielsprache enthalten und den zu erzeugenden Code bspw. in Form eines Abstract Syntax Tree beinhalten. Um aus dem Syntaxbaum Quellcode erzeugen können ist zusätzlich ein *Renderer* zu implementieren. Gegenüber der Verwendung von Templates bietet dieser Ansatz folgende Vorteile:

- Formatierung des erzeugten Codes über Parameter des *Renderers*, bspw. Einrückungstiefe, Position von Klammern (auf neuer Zeile), …
- Optimierung des zu erzeugenden Codes durch Analyse des Abstract Syntax Tree
- Trennung von Syntax (Renderer) und Semantik (Sprachenmodell)
- Implementierung einer neuen Zielsprache nur durch Anpassung des Renders

Ein Nachteil ist der wesentlich höhere Implementierungsaufwand.

3 Installation und Verwendung

Um den Generator nutzen zu können müssen noch die nötigen Abhängigkeiten installiert werden. Diese sind in der Datei `requirements.txt` im Stammverzeichnis des Projektes festgehalten und können mit Hilfe des Python Pakettools *pip* wie folgt installiert werden: `pip install -r requirements.txt`.

Es werden zwei Kommandozeilenparameter vom Codegenerator erwartet, erstens eine URL welche den Ort der XSD-Datei angibt und zweitens den lokalen Pfad an dem das generierte Java Package angelegt werden sollen. Weiterhin existieren noch optionale Parameter zur Ausgabe von Debuginformationen (`-d/--debug`) oder—für die spätere Erweiterung gedacht—die Angabe der zu verwendenden Zielsprache mittels `-l/--lang` (Standard ist Java).

Die Hilfe kann mittels `-h/--help` ausgegeben werden, bei falscher Verwendung wird diese sowie ein Nutzungshinweis aber automatisch angezeigt.

4 Fazit

Das Projekt diente hauptsächlich dazu sich mit dem Thema Codegenerierung in Python zu beschäftigen und Erfahrung mit Template-Engines und XML Bibliotheken in dieser Sprache zu sammeln. Der entstandene Generator kann auch mit überschaubaren Änderungen so erweitert werden das man ihn produktiv einsetzen kann.

4.1 Erweiterungsmöglichkeiten

Wie in Abschnitt 1.1 (S. 3) erwähnt ist der Einsatzzweck eines solchen Generators optimalerweise als Teil eines Generatorsystems zu sehen, welches aus einer kompletten Beschreibung eines RESTful Webservice in Form von WADL- und XSD-Dateien eine Client-Bibliothek erzeugt. Mit einem solchen Generatorsystem kann man bei Änderungen an der Webservice-API eine neue Client-Bibliothek generieren die den Zugriff auf alle Ressourcen erlaubt ohne das der Nutzer sich mit Details der Kommunikation auseinandersetzen muss.

Um das erzeugte Java Paket sinnvoll nutzen zu können sollten noch Deserialisierungsmethoden generiert werden, die die Typklassen direkt aus der XML-Darstellung initialisieren. Zustätzlich sollten noch mehrere Aspekte der XML Schemadefinition bei der Generierung berücksichtigt werden, wie anonyme Typdefinitionen, die man als z.B. Innere Klassen generieren könnte.

```
<xsd:element name="bar">
  <xsd:complexType>
    <xsd:sequence>
      <xsd:element name="foo" type="xsd:decimal"/>
    </xsd:sequence>
    <xsd:attribute name="baz" type="xsd:boolean" />
  </xsd:complexType>
</xsd:element>
```

```
class Bar {
    class Anonym {
        private Double foo;
        private Boolean baz;
        ...
        public String toXML() {
            return "<foo>"+this.foo+"</foo>"
        }
    }

    private Anonym value;
    ...
    public String toXML() {
        "<bar␣" + "baz=" + value.baz + "/>" + value.toXML() + "</bar>"
    }
}
```

Die Implementierung zusätzlicher Zielsprachen durch die Erstellung weiterer Templates ist vorgesehen, ebenfalls ist ein Kommandozeilenparameter `-l/--lang LANGUAGE` bereits implementiert unter dessen Berücksichtigung das `render` Modul den entsprechenden Renderer auswählt.

5 Appendix

Glossar

Restful Als *RESTful* bezeichnet man einen Webservice der den Prinzipien von REST entspricht. 3, 8, *siehe* Rest

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 [Ber+98]).. 10

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 [Ber+98]).. 8, 10, *siehe* Uri

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 [Ber+98]).. 10, *siehe* Uri

Wadl *Web Application Description Language* ist eine maschinenlesbare Beschreibung einer HTTP-basierten Webanwendung. 3, 8, *siehe* Xml

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 [W3C08] . 3, 5, 6

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. 3–5, 8, *siehe* Xml

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, beispielsweise formatierende Element wie etwa Klammern werden häufig weggelassen . 7

Template-Engine Eine *Template-Engine* ersetzt markierte Bereiche in einer Template-Datei (i. Allg. Textdateien) nach vorgegebenen Regeln . 7

Literatur

[Beh14] Stefan Behnel. *lxml*. 2014. URL: http://lxml.de/ (besucht am 31. 05. 2014).

[Ber+98] Tim Berners-Lee u. a. *Uniform Resource Identifiers (URI): Generic Syntax*. Aug. 1998. URL: http://www.ietf.org/rfc/rfc2396.txt (besucht am 05. 26. 2014).

[CE00] K. Czarnecki und U. Eisenecker. *Generative programming: methods, tools, and applications*. Addison Wesley, 2000. ISBN: 9780201309775.

[Fac14] Facebook. *Facebook-API XSD*. 2014. URL: http://api.facebook.com/1.0/facebook.xsd (besucht am 26. 05. 2014).

[mak14] mako. *Python Mako Template-Engine*. 2014. URL: http://www.makotemplates.org/ (besucht am 31. 05. 2014).

[W3C04] W3C. *XML Schema Part 0: Primer Second Edition*. 28. Okt. 2004. URL: http://www.w3.org/TR/xmlschema-0/ (besucht am 31. 05. 2014).

[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).