Generierung von Java Klassen aus einer XML Schemabeschreibung (XSD)

Generative Software Development – Projektvorstellung

Andreas Linz

Universität Leipzig

1. Juni 2014

Inhaltsverzeichnis

- 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

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 Serialisierung- und Deserialisierungsmethoden[2]

[2] Werden derzeit nicht generiert.

- 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]

```
<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>
```

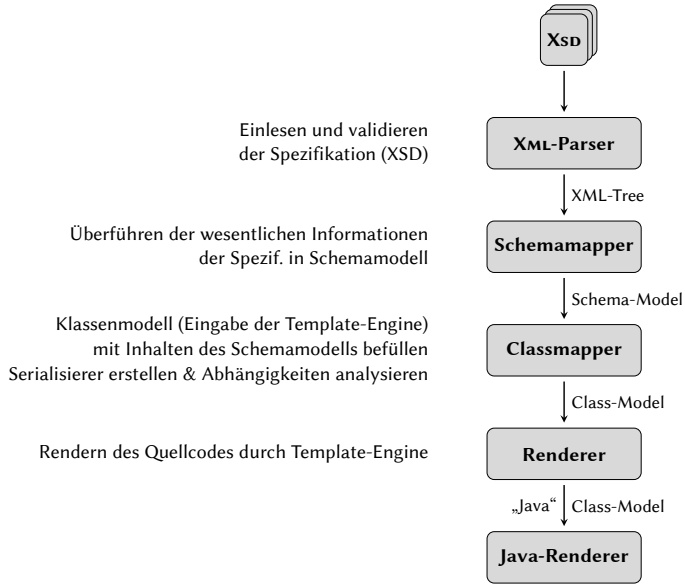
Listing 3: Beispiel für einen strukturierten Schematyp aus [Fac14]

```
<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>
```

- 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



- 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

Listing 4: Beispieltemplate

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

Fazit

Dies hier ist ein Blindtext zum Testen von Textausgaben. Wer diesen Text liest, ist selbst schuld. Der Text gibt lediglich den Grauwert der Schrift an. Ist das wirklich so? Ist es gleichgültig, ob ich schreibe: Dies ist ein Blindtext oder Huardest gefburn? Kjift -- mitnichten! Ein Blindtext bietet mir wichtige Informationen. An ihm messe ich die Lesbarkeit einer Schrift, ihre Anmutung, wie harmonisch die Figuren zueinander stehen und prüfe, wie breit oder schmal sie läuft. Ein Blindtext sollte möglichst viele verschiedene Buchstaben enthalten und in der Originalsprache gesetzt sein. Er muß keinen Sinn ergeben, sollte aber lesbar sein. Fremdsprachige Texte wie Lorem ipsum dienen nicht dem eigentlichen Zweck, da sie eine falsche Anmutung vermitteln.