Cheatsheet für Patterns in generativen Schemasprachen

Anne Brüggemann-Klein, TUM

Zusammenfassung

In diesem Cheatsheet geht es um die vier Patterns, nach denen ein generatives Schema für XML-Dokumente aufgebaut sein kann, nämlich Salami Slice, Russian Doll, Venetian Blind und Garden of Eden. Es gibt zunächst eine konzeptuelle Erklärung. Dann wird für konkrete Schemasprachen diskutiert, wie die Patterns in dieser Schemasprache konkret umgesetzt werden. In der ersten Version dieses Cheatsheets betrachten wir schwerpunktmäßig die Schemasprache XML Schema, aber auch XML DTD. In einer weiteren Version wird Relax NG folgen.

Das Cheatsheet ist als Ergänzung zum Skript "Elektronisches Publizieren: Document Engineering und das World-Wide Web" gedacht und wird in einer Neuauflage in das Skript integriert.

Inhaltsverzeichnis

Elementdeklarationen und Typdefinitionen

Die konzeptionelle Sicht

Eine *Schemasprache* für XML sieht Mechanismen vor, um Schemata für XML-Dokumente zu definieren. Diese Mechanismen betrachten wir hier auf einer konzeptionellen und auf einer syntaktischen Ebene. Beispiele für Schemasprachen sind XML DTD, XML Schema, Schematron und Relax NG. Ein konkretes *Schema* aus einer Schemasprache definiert eine Menge von konformen oder validen XML-Dokumenten, den Instanzen. Von der Dokumentenseite her gesehen kann ein konkretes XML-Dokument gegenüber einem konkreten Schema gegenüber *valide* sein. Ein validierender Parser, auch XML-Prozessor genannt, kann überprüfen, ob ein XML-Dokument die Vorschriften eines Schemas erfüllt, ob es also dem Schema gegenüber valide ist bzw. eine *Instanz* des Schemas bildet. Wir halten also die drei Ebenen Schemasprache, Schema und Instanz auseinander: Ein Schema ist in einer Schemasprache geschrieben und hat eine Menge von XML-Dokumenten als Instanzen.

Eine Schemasprache kann *generativ* oder *deklarativ* sein. Der Mechanismus einer generativen Schemasprache erlaubt es, systematisch die validen Instanzen eiens Schemas aus dem Schema heraus zu erzeugen. Generative Schemasprachen funktionieren wie Grammatiken in der Theorie der Formalen Sprachen. Generative Schemasprachen sind XML DTD, XML Schema und Relax NG. Die Ausnahme bildet Schematron, die einzige deklarative Schemasprache in unserer Liste. Ein Schema gemäß Schematron gibt Regeln an für die Bestandteile eines konformen XML-Dokuments und für deren Beziehungen untereinander. Aus diesen Regeln kann man Instanzen nicht systematisch erzeugen. Man kann lediglich für ein konkretes XML-Dokument überprüfen, wieder mit einem validierenden Parser, ob es die Regeln erfüllt.

Eine generative Schemasprache besteht aus Elementdeklarationen und Typdefinitionen. Eine *Elementdeklaration* bezieht sich immer auf einen *Elementnamen* und definiert, welchen *Typ* die Elemente mit diesem Namen habe sollen. Der Typ für ein Element legt beispielsweise fest,

1. welche Unterelemente ein Element haben kann und von welchem Typ diese Unterelemente sind.

2. ob das Element nur Text enthalten soll (*text content*) oder nur Unterelemente (*element content*) oder eine Mischform (*mixed content*).

3. welche Attribute das Element enthalten darf oder muss.

Diese Festlegung trifft eine Elementdeklaration über eine Typdefinition.

Eine *Typdefinition* für ein Element von einem bestimmten Namen legt gegebenenfalls in einem sogenannten *Inhaltsmodell* fest, welche Unterelemente vorkommen dürfen oder müssen und von welchem Typ diese Unterelemente jeweils sein sollen; dies trifft auf die Fälle *element content* und *mixed content* zu. Eine Typdefinition kann in einem Inhaltsmodell also auch selbst wieder Elementdeklarationen beinhalten. Im Fall von *text content* legt die Typdefinition einen Datentyp für den Textinhalt fest.

Darüber hinaus kann eine Typdefinition für ein Element von einem bestimmten Namen festlegen, welche Attribute das Element tragen kann oder muss. Bei der Festlegung eines Attributs wird der Name und der Datentyp des Attributs spezifiziert.

Die syntaktische Ausprägung in XML Schema

Eine Elementdeklaration in XML Schema erfolgt nach dem Muster[1]

```
<xs:element name="..."/>
```

Das Attribut `name` von `xs:element` enthält dabei den Namen des zu deklarierenden Elements. Konkrete Instanzen dieser Deklaration sind also Elemente des angegebenen Namens.

Eine Typdefinition ist in XML Schema durch das Element `xs:complexType` oder `xs:simpleType` gegeben. Typendefinitionen auf der obersten Ebene eines Schemas müssen dem Typen über das Attribut `name` einen Namen geben. Über diesen Namen kann der Typ referenziert werden, entweder in dem Schema, in dem er definiert ist, selbst oder in einem anderen Schema, das das Schema mit der Typdefinition importiert. Die Namen der vordefinierten Datentypen wie `xs:string` oder `xs:date` können immer referenziert werden, als wenn ihre Definitionen automatisch importiert wären.

Wie kann eine Elementdeklaration in XML Schema einen Typen für das zu deklarierende Element festlegen? Dazu gibt es zwei Möglichkeiten: Entweder wird eine benannte Typdefinition über das Attribut `type` von `xs:element` referenziert. Diese Methode wird immer genutzt, wenn vordefinierte Datentypen wie `xs:integer` referenziert werden, kann aber auch auf selbstdefinierte und benannte Typen angewendet werden. Eine Elementdeklaration kann aber auch als Inhalt des Elements `xs:element` eine Typdefinition lokal enthalten, die dann keinen Namen haben darf, also anonym sein muss.

Wie kann eine Typdefinition für einen komplexen Typ in ihrem Inhaltsmodell über Kindelemente und deren Typ sprechen? Auch hier gibt es zwei Möglichkeiten: Entweder wird das Element lokal innerhalb der Typdefinition deklariert wie oben besprochen. Oder es wird eine Elementdeklaration, die auf oberster Ebene des Schemas erfolgt ist, referenziert. Dazu dient in XML Schema ebenfalls das Element `xs:element`, allerdings nicht mit dem Attribut `name` versehen sondern mit dem Attribut `ref`. Eine Referenz auf eine solche Elementdeklaration enthält dann natürlich nicht noch einmal eine eigene Typdefinition sondern übernimmt die Typdefinition aus der referenzierten Deklaration.

Der Vollständigkeit halber noch ein paar Worte zur Handhabung von Attributen in XML Schema: Der am meisten verwendete Mechanismus fügt in die Definition eines komplexen Typs nach der Festlegung des Inhaltsmodells die Attributdeklarationen ein, die nach dem folgenden Schema:

```
<xs:attribute name="..." type="..." use="..."/>
```

Dabei gibt `name` den Namen des Attributs an, `type` den Typ und `use` die Vorschrift zur Verwendung (optional, verpflichtend).

Wir schauen uns nun ein konkretes annotiertes Schema aus der Sprache XML Schema an:

[1] Hier und in den folgenden Beispielen ist das Präfix xs immer an den Namensraum http://www.w3.org/2001/XMLSchema von XML Schema gebunden.

Abbildung 1. Ein annotiertes Schema gemäß XML Schema nach dem Muster Salami Slice

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <!-- Deklaration von Element gedicht (global) -->
    <xs:element name="gedicht">
        <!-- Typdefinition für gedicht (lokal) -->
        <xs:complexType>
            <xs:sequence>
                <!-- Referenz der global deklarierten Elemente
                     autor und titel -->
                <xs:element ref="autor"/>
                <xs:element ref="titel"/>
            </xs:sequence>
            <!-- Typdeklaration als Teil der Typdefinition -->
            <xs:attribute name="jahr" type="xs:integer" use="optional"/>
        </xs:complexType>
    </xs:element>
    <!-- Deklaration von Element autor (global) -->
    <xs:element name="autor">
        <!-- Typdefinition für autor (lokal) -->
        <xs:complexType>
            <xs:sequence>
                <!-- Referenz der global deklarierten Elemente
                     vorname und nachname -->
                <xs:element ref="vorname"/>
                <xs:element ref="nachname"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
    <!-- Deklaration der Elemente titel, vorname und nachname (global)
         mit Referenz jeweils auf den benannte vordefinierten
         Typ xs:string -->
    <xs:element name="titel" type="xs:string"/>
    <xs:element name="vorname" type="xs:string"/>
    <xs:element name="nachname" type="xs:string"/>
</xs:schema>
```

Eine Instanz dieses Schemas ist das folgende nur partiell wiedergegebene Rilke-Gedicht, das uns auch später noch als Beispiel dienen wird.

Abbildung 2. Ein XML-Dokument als Instanz verschiedener Schemata

```
<gedicht>
    <autor>
        <vorname>Rainer Maria</vorname>
        <nachname>Rilke</nachname>
    </autor>
    <titel>Als ich die Universität bezog</titel>
</gedicht>
```

Patterns für generative Schemasprachen

Wir haben gesehen, dass generative Schemasprachen Elementdeklarationen und Typdefinitionen enthalten können. Diese beiden Komponenten können *global* auf der obersten Ebene des Schemas ange-

ordnet sein oder *lokal* in andere Komponenten eingebettet werden. Im Falle von XML Schema befinden sich die globalen Komponenten auf der Kindebene von dem äußersten Container-Element für ein Schema, `xs:schema`.

Globale Komponenten haben einen Namen, über den sie referenziert und wiederverwendet werden können. Im Falle von XML Schema ist der Name sowohl in Elementdeklarationen als auch in Typdefinitionen über das Attribut `name` angegeben. Elementdeklarationen haben immer einen Namen, auch wenn sie lokal vorkommen, während Typdefinitionen nur als globale Komponenten einen Namen haben und bei lokalem Vorkommen anonym sind.

Lokale Elementdeklarationen kommen im Inhaltsmodell von Typdefinitionen vor. Im Falle von XML Schema sind die lokalen Elementdeklarationen nur an ihrer inneren Position als lokal erkennbar. Sie sind genauso wie die globalen Elementdeklarationen über das Element `xs:element` mit Attribut `name` gegeben. Anstatt in einem Inhaltsmodell einer Typdefinition ein Element lokal zu deklarieren, kann auch eine globale Elementdeklaration referenziert werden. In XML Schema passiert das auch über das Element `xs:element`, in dem aber das Attribut `name` durch das Attribut `ref` ersetzt ist.

Lokale Typdefinitionen kommen innerhalb von Elementdeklarationen vor, als Inhalt von einem Element `xs:element` mit Attribut `name`. Die Elementdeklaration kann dabei lokal oder global sein. Auch Typdefinitionen können referenziert werden. Dafür müssen sie global definiert sein und damit einen Namen haben. Die Referenzierung erfolgt dann in der Elementdeklaration über das Attribut `type` und den Namen des Typs.

In dem Beispielschema Abbildung 1, „Ein annotiertes Schema gemäß XML Schema nach dem Muster Salami Slice“ kommen globale und lokale Typdefinitionen vor sowie Referenzen auf globale Elementedeklarationen und vordefinierte Datentypen.

Eine generative Schemasprache kann also lokale und globale Elementdeklarationen sowie lokale und globale Typdefinitionen vorsehen. In XML Schema sind alle diese Möglichkeiten gegeben. In XML DTD dagegen gibt es nur globale Elementdeklarationen und lokale Typdefinitionen (mit Ausnahme der Referenzierung von vordefinierten Datentypen wie CDATA, PCDATA oder ID). Jedes Schema benötigt mindestens eine globale Elementdeklaration für das Einstiegselement von Instanzen. Außerdem referenziert jedes Schema vordefinierte Datentypen oder aus anderen Schemata importierte Datentypen.

Abgesehen von den oben erwähnten Ausnahmen kann ein Schema so strukturiert sein, dass es nur globale oder nur lokale Elementdeklarationen verwendet und nur globale oder nur lokale Typdefinitionen. Damit ergeben sich vier Kombinationsmöglichkeiten oder *Muster*, auch *Patterns* genannt, für Schemata:

- *Salami Slice:* Globale Elementdeklarationen und lokale Typdefinitionen. Schemata nach XML DTD sind immer nach diesem Muster aufgebaut.
- *Venetian Blind*: Lokale Elementdeklarationen und globale Typdefinitionen. Das duale Gegenstück zu Salami Slice.
- *Russian Doll*: Lokale Elementdeklarationen und lokale Typdefinitionen.
- *Garden of Eden*: Globale Elementdeklarationen und globale Typdefinitionen.

Das Beispiel in Abbildung 1, „Ein annotiertes Schema gemäß XML Schema nach dem Muster Salami Slice“ enthält nur globale Elementdeklarationen und lokale Typdefinitionen, mit Ausnahme von implizit vorliegenden und referenzierten globalen Typdefinitionen für vordefinierte Datentypen. Damit entspricht das Beispiel dem Muster Salami Slice. Für die anderen drei Muster gibt es weiter unten Beispiele. Für jedes der vier Schemata ist das XML-Dokument in Abbildung 2, „Ein XML-Dokument als Instanz verschiedener Schemata“ eine Instanz.

Das Schema in Abbildung 3, „Ein Schema nach dem Muster Russian Doll“ enthält außer einer globalen Elementdeklaration für das Einstiegselement `gedicht` und den Referenzen auf den vordefinierte

Datentyp `xs:string` nur lokale Typdefinitionen und lokale Elementdeklarationen. Das Schema ist also nach dem Muster Russian Doll strukturiert.

Abbildung 3. Ein Schema nach dem Muster Russian Doll

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="gedicht">
        <xs:complexType>
            <xs:sequence>
                <xs:element name="autor">
                    <xs:complexType>
                        <xs:sequence>
                            <xs:element
                                name="vorname" type="xs:string"/>
                            <xs:element
                                name="nachname" type="xs:string"/>
                        </xs:sequence>
                    </xs:complexType>
                </xs:element>
                <xs:element name="titel" type="xs:string"/>
            </xs:sequence>
        </xs:complexType>
    </xs:element>
</xs:schema>
```

Das Schema in Abbildung 4, „Ein Schema nach dem Muster Venetian Blind“ enthält außer einer globalen Elementdeklaration für das Einstiegselement `gedicht`nur globale und referenzierte Typdefinitionen und lokale Elementdeklarationen. Das Schema ist also nach dem Muster Venetian Blind strukturiert.

Abbildung 4. Ein Schema nach dem Muster Venetian Blind

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="gedicht" type="gedichtTyp"/>
    <xs:complexType name="gedichtTyp">
        <xs:sequence>
            <xs:element name="autor" type="autorTyp"/>
            <xs:element name="titel" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="autorTyp">
        <xs:sequence>
            <xs:element name="vorname" type="xs:string"/>
            <xs:element name="nachname" type="xs:string"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Das Schema in Abbildung 5, „Ein Schema nach dem Muster Garden of Eden“ enthält außer einer globalen Elementdeklaration für das Einstiegselement `gedicht` und den Referenzen auf den vordefinierte Datentyp `xs:string` nur lokale Typdefinitionen und lokale Elementdeklarationen. Das Schema ist also nach dem Muster Garden of Eden strukturiert.

Abbildung 5. Ein Schema nach dem Muster Garden of Eden

```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema">
    <xs:element name="gedicht" type="gedichtTyp"/>
    <xs:element name="autor" type="autorTyp"/>
    <xs:element name="titel" type="xs:string"/>
    <xs:element name="vorname" type="xs:string"/>
    <xs:element name="nachname" type="xs:string"/>
    <xs:complexType name="gedichtTyp">
        <xs:sequence>
            <xs:element ref="autor"/>
            <xs:element ref="titel"/>
        </xs:sequence>
    </xs:complexType>
    <xs:complexType name="autorTyp">
        <xs:sequence>
            <xs:element ref="vorname"/>
            <xs:element ref="nachname"/>
        </xs:sequence>
    </xs:complexType>
</xs:schema>
```

Zusammenfassung

In diesem Cheatsheet werden die vier Muster oder Patterns Salami Slice, Venetian Blind, Russian Doll und Garden of Eden, nach denen Schemata strukturiert sein können, konzeptuell definiert und anhand der Schemasprache XML Schema illustriert. Studierende sollen damit in die Lage versetzt werden, an einem konkreten Schema zu erkennen, nach welchem Muster es aufgebaut ist. Davon unberührt bleibt die Frage, welche Eigenschaften, Vor- und Nachteile die verschiedenen Patterns haben und wie sie in Bezug auf Beschreibungsmächtigkeit miteinander verglichen werden können. Eine spätere Fassung wird genauer auf die Schemasprache XML DTD eingehen und die Schemasprache Relax NG behandeln.