

Belegarbeit

Transformation eines XML-Dokuments mittels EditiX und Python (lxml)

Modul Internettechnologie

Prof. Dr.-Ing. Andreas Pretschner
HTWK Leipzig

Maurice Götze, Felix Kühn
maurice.goetze@stud.htwk-leipzig.de
felix.kuehn@stud.htwk-leipzig.de

20. April 2022

1 Einleitung

Dieser Beleg zeigt die Vorgehensweise beim Erstellen und der Transformation von XML-Dokumenten. Für den ersten Teil der Ausarbeitung wird der XML-Editor EditiX 2020 verwendet. Darin wird im ersten Schritt ein XML-Dokument erstellt, welches mit Beispielen von Open-Source-Werkzeugen gefüllt wird. Dieses Dokument wird anschließend gegen ein XSD-Schema geprüft, welches die Strukturvorgabe und Soll-Datentypen der Elemente enthält. Im letzten Schritt werden die Rohdaten aus dem XML-Dokument in HTML und PDF Dateien transformiert, die für die anschauliche Visualisierung der Daten genutzt werden können.

Der zweite Teil des Belegs zeigt die Herangehensweise an Validierung und Transformation mittels der Programmiersprache Python. Das Paket *lxml* ermöglicht das Parsen und die Verarbeitung von XML-Daten auf der Konsole. Im Gegensatz zum Editor EditiX kann diese Methode zur automatischen Erstellung von Dokumenten z.B. in Serverumgebungen ohne grafische Benutzeroberfläche genutzt werden.

2 Aufbau des XML-Dokuments

Im XML-Dokument sollen verschiedene Open-Source Werkzeuge aufgelistet werden. Jedes Werkzeug ist also ein Element mit dem Namen *tool*. Alle Werkzeuge werden dem Root-Element zugeordnet, welches hier den Namen *tools* trägt und in Abbildung 2 blau dargestellt ist.

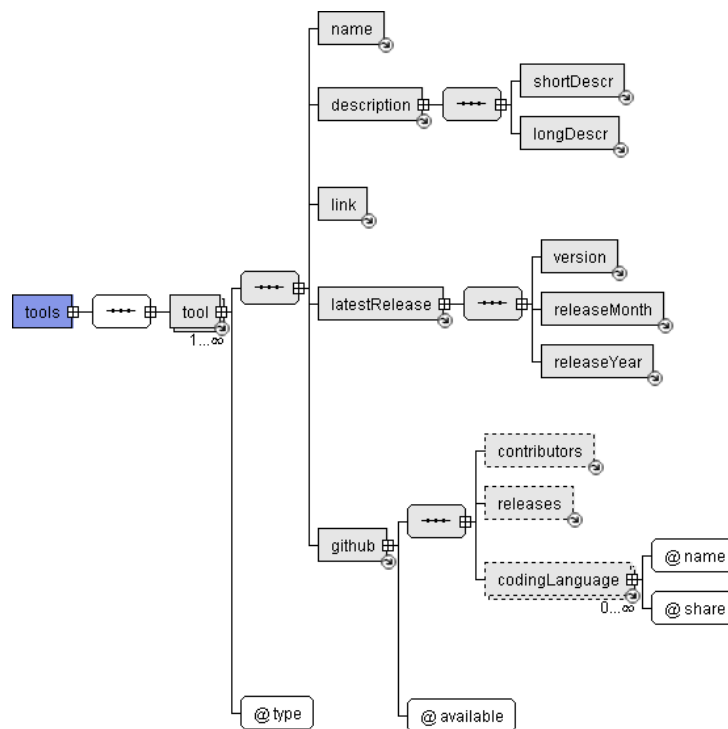


Abbildung 2.1: Root-Element „tools“ mit allen Elementen und Attributen (generiert aus Editix)

Zur Beschreibung des Werkzeuges wird dessen Name, eine Kurzbeschreibung der Funktionalität und ein Link zur Webseite angegeben. Zusätzlich wird die letzte Veröffentlichung aufgezeigt, welche mit deren Nummer und Veröffentlichungsmonat angegeben wird. Sollte das Werkzeug auch auf einer einzelnen Github-Seite veröffentlicht sein, wird dieses ebenfalls angegeben. Die Elemente und Attribute der einzelnen Pfade sind in der Übersicht in Abbildung 2 aufgezeigt. Entwickelt wurden die folgenden Skripte und Dateien in den Entwicklungsumgebungen PyCharm in der Version 2021.3.3 (Community Edition) und EditiX Professional 2020.

3 Validierung

Für die Validierung der XML-Struktur wurde eine Schemadatei (XSD) in EditiX erzeugt. Dieses Schema garantiert, dass alle Daten, die im XML-Dokument enthalten sind, korrekt eingegeben bzw. erzeugt wurden. Dazu gehört die Angabe der Typen für jedes Element und jedes Attribut. Neben der Typangabe ist es ebenfalls wichtig, die Beschränkungen hinsichtlich Häufigkeit eines Elements anzugeben. Die Beschränkungen werden im Quelltext 3.1 beispielhaft für das Element *Github* eines jeden Open-Source-Werkzeuges dargestellt.

```
1 <xs:element name="github">
2   <xs:complexType mixed="true">
3     <xs:sequence>
4       <xs:element ref="contributors" minOccurs="0"/>
5       <xs:element ref="releases" minOccurs="0"/>
6       <xs:element ref="codingLanguage" minOccurs="0" maxOccurs="unbounded"/>
7     </xs:sequence>
8     <xs:attribute name="available" type="xs:string" use="required"/>
9   </xs:complexType>
10 </xs:element>
```

Quelltext 3.1: Beispielement *Github* im XSD-Schema

Während das Attribut *available* in jedem Github-Element zwingend angegeben werden muss (required), sind die Angaben zu *contributors*, *releases* und *CodingLanguages* optional (*minOccurs* = "0"). Mit diesem Schema kann das XML-Dokument validiert werden. Dazu wird die in EditiX enthaltene Funktion *check* verwendet. Als Ausgabe liefert EditiX „*Your Document is correct.*“, was darauf schließen lässt, dass alle Elemente und Attribute in der richtigen Struktur angegeben sind. Die automatische Ausleitung der Dokumentation aus EditiX ist dem Beleg angehängt siehe Seite 11.

4 Transformation

Die Transformation des XML-Dokuments erfolgt mit dem Anwenden einer XSL-Transformation. Für die in diesem Beleg betrachteten Transformation (zu HTML und zu PDF) wurde jeweils eine Transformationsdatei erstellt. Die Ergebnisse beider Transformationen sind im Belegverzeichnis unter den Dateinamen *result.html* und *result.pdf* aufgeführt.

4.1 HTML

Für die Transformation in eine Webseite muss in der XSL-Datei als erstes die grundlegende Struktur einer Webseite mit den entsprechenden Tags angegeben werden (`<title>`, `<header>`, `<body>`). Innerhalb dieser Struktur wird dann die Tabelle aufgebaut. Als erstes wird der Tabellenkopf mit statischem Text beschrieben und dann zeilenweise die Daten aus dem XML-Dokument verwendet. Durchgeführt wird dies mit dem Aufrufen von Templates, die für jedes XML-Element den gewünschten Wert auslesen und in die Tabelle eintragen. Das Ergebnis der HTML Transformation ist eine Webseite, welche alle Daten des XML-Dokuments in Tabellenform darstellt.

4.2 FO-PDF

Das Vorgehen bei der Transformation zu PDF ähnelt dem zu HTML. Auch hier wird als Erstes die grundlegende Struktur eines FO-Objekts erstellt und erste Konfigurationen der Seite getroffen (Querformat, Seitenabstände, Textgröße). Im Anschluss wird auch bei dieser Transformation eine Tabelle erstellt, die dann durch das Aufrufen der Templates mit Daten gefüllt wird. EditiX transformiert das XML-Dokument in ein FO-Objekt, welches dann zum Erstellen des PDF-Dokuments verwendet wird. Das Ergebnis der PDF Transformation ist ein PDF-Dokument, welches alle Daten des XML-Dokuments in Tabellenform darstellt.

5 Skriptentwurf

Ziel dieses Abschnitts ist der Entwurf einer konsolenbasierten Anwendung, die XML-Dokumente validieren und transformieren kann. Dazu wird die Programmiersprache Python mit dem Erweiterungspaket *lxml* verwendet. Die Dokumentation zum Paket kann unter <https://lxml.de/> gefunden werden. Nachdem es z.B. mit *pip install lxml* installiert wurde, kann der Elementparser mit dem Namen *etree* zu einem Skript hinzugefügt werden, was der Quelltext 5.1 zeigt. Die folgenden Skripte wurden in der Entwicklungsumgebung PyCharm in der Version 2021.3.3 (Community Edition) erstellt.

```
1 from lxml import etree #documentation on https://lxml.de/
    validation.html
```

Quelltext 5.1: Importieren von lxml

5.1 lxml Validierung

Im ersten Schritt werden die Dokumente eingelesen. Dazu wurde die Funktion *readFile* erstellt, welche die Dokumente als Strings aus den Dateien einliest, und Tabulatoren sowie Zeilenumbrüche entfernt. Diese Funktion liefert eine Zeichenkette, die nacheinander die öffnenden und schließenden Tags der XML-Datenstruktur enthält. Anschließend wird die Schema-Datei mit dem Befehl *etree.XMLSchema* geparsed, wie im Quelltext 5.2 in Zeile 7 zu sehen ist.

```
1 def readSchema():
2     #read documents from files
3     tools_schema = readFile('tools_schema.xsd')
4     tools_xml = readFile('tools.xml')
5     tools_xml_invalid = readFile('tools_invalid.xml')
6     #parse xml schema
7     tools_xmlschema = etree.XMLSchema(etree.parse(tools_schema
8     ))
9     #validate xmls against schema
10    validation(tools_xmlschema, 'tools_schema.xsd', tools_xml,
11    'tools.xml')
```

```
10 validation(tools_xmlschema, 'tools_schema.xsd',  
            tools_xml_invalid, 'tools_invalid.xml')
```

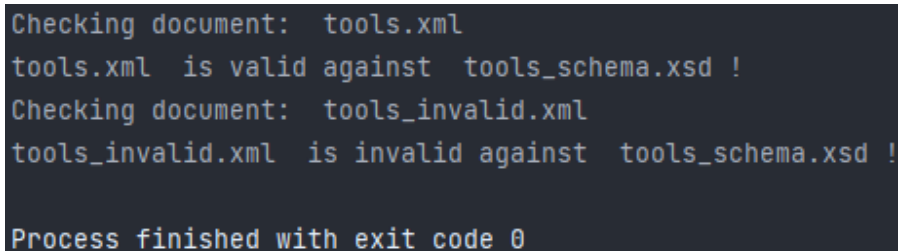
Quelltext 5.2: Funktion zum Einlesen und Parsen der Dokumente in `validate.py`

Als nächstes werden die eingelesenen XML-Dokumente zusammen mit dem zugehörigen Schema an die Funktion *validation* übergeben. Um eine erfolgreiche und eine fehlgeschlagene Validierung zu zeigen, wurde eine Kopie von der im Beleg verwendeten XML-Struktur *tools.xml* erstellt und einige notwendigen Elemente gelöscht. Dadurch entstehen zwei Funktionsaufrufe in Zeile 9 und 10 im Quelltext 5.2.

```
1 def validation(schema, schema_name, xml, filename):  
2     print("Checking document: ", filename)  
3     val = etree.parse(xml) #parsing each element of xml  
4     schema.validate(val) #validate the parsed xml against the  
        schema  
5     #print(schema(val))  
6     if schema(val):  
7         #schema(val) returns true --> document is valid  
8         print(filename, " is valid against ", schema_name, "!"  
            )  
9     else:  
10        # schema(val) returns false --> document is invalid  
11        print(filename, " is invalid against ", schema_name, "  
            !")
```

Quelltext 5.3: Funktion zur Validierung in `validate.py`

In der Funktion Quelltext 5.3 wird im Wesentlichen der Befehl *schema.validate(xml)* aufgerufen, welcher *True* liefert, wenn das Dokument korrekt im Bezug zum Schema ist und *False* wenn nicht.



```
Checking document: tools.xml  
tools.xml is valid against tools_schema.xsd !  
Checking document: tools_invalid.xml  
tools_invalid.xml is invalid against tools_schema.xsd !  
  
Process finished with exit code 0
```

Abbildung 5.1: Ausgabe der Funktion *validate.py*

In Abbildung 5.1 wird das Ergebnis des Skriptdurchlaufs auf einer Konsole gezeigt. Wie zu erwarten, ist das XML-Dokument *tools.xml* korrekt, während das manuell veränderte XML-Dokument *tools_invalid.xml* Fehler in Bezug auf das XML-Schema enthält.

5.2 lxml Transformation

Für die Transformation von XML-Dokumenten kann das Paket *lxml* ebenfalls verwendet werden. Dazu werden in Quelltext 5.4 die XML- und XSL-Dokumente eingelesen und anschließend mit dem Befehl *transform* in ein neues Objekt gewandelt. Dieses Objekt wird danach als Zeichenfolge im utf-8-Format ausgegeben und als HTML-Datei gespeichert.

```
1 xslt = etree.parse("tools_html.xsl") #parse xsl file
2 xml = etree.parse("tools.xml") #parse xml file
3 def transformToHtml():
4     transform = etree.XSLT(xslt)
5     newdom = transform(xml)
6     print(etree.tostring(newdom, pretty_print=True))
7     infile = (etree.tostring(newdom, pretty_print=True)).
        decode("utf-8")
8     outfile = open("result_html.html", 'w+')
9     outfile.write(infile)
```

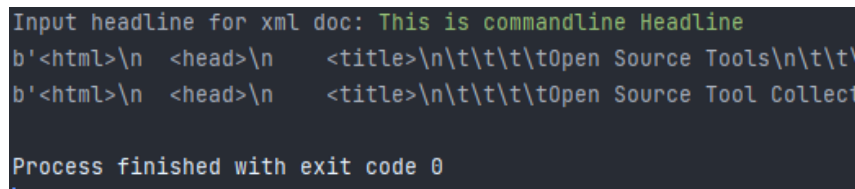
Quelltext 5.4: Transformieren des XML-Dokuments zu HTML mit transform.py

Das Resultat dieser Transformation ist identisch mit dem Resultat des aus EditiX generierten Dokuments. Um die Vorteile der automatischen Transformation mittels eines Skripts weiter auszunutzen, kann der Befehl *transform* mit weiteren Übergabewerten bestückt werden. Damit lassen sich Parameter der Transformation aus dem Skript heraus steuern, ohne das XSL-Dokument verändern zu müssen. Im Beispiel in Quelltext 5.5 wird der Parameter *text* an die Transformation übergeben.

```
1 xslt_par = etree.parse("tools_html_param.xml")
2 def transformToHtmlParam():
3     transform = etree.XSLT(xslt_par)
4     text = input("Input headline for xml doc: ")
5     newdom = transform(xml, var1=etree.XSLT.strparam(text))
6     print(etree.tostring(newdom, pretty_print=True))
```

Quelltext 5.5: Transformieren des XML-Dokuments inklusive Parameterangabe mit transform.py

Die Webseite ändert damit in Abhängigkeit von der Benutzereingabe den Text in der Tabellenüberschrift. Dazu wurde in der Transformation die Zeile `<xsl:value-of select="$var1"/>` an entsprechender Stelle der Überschrift hinzugefügt.



```
Input headline for xml doc: This is commandline Headline
b'<html>\n  <head>\n    <title>\n\t\t\t\t\tOpen Source Tools\n\t\t\t\t\tOpen Source Tool Collect
Process finished with exit code 0
```

Abbildung 5.2: Eingabe der neuen Überschrift auf der Konsole

Open Source Tool Collection

This is commandline Headline

Tool Name	Type	Description
OpenCV	Library	OpenCV is a library that includes computer vision algorithms

Abbildung 5.3: Ergebnis der parameterabhängigen Transformation (Ausschnitt aus der Webseite)

In Abbildung 5.2 und Abbildung 5.3 wird die parameterabhängige Transformation verdeutlicht. Dieses Vorgehen kann in ähnlicher Weise vor allem dort eingesetzt werden, wo Transformationen in Abhängigkeit von äußeren Parametern beeinflusst werden, die nicht innerhalb des XSL-Dokuments erfasst werden können.

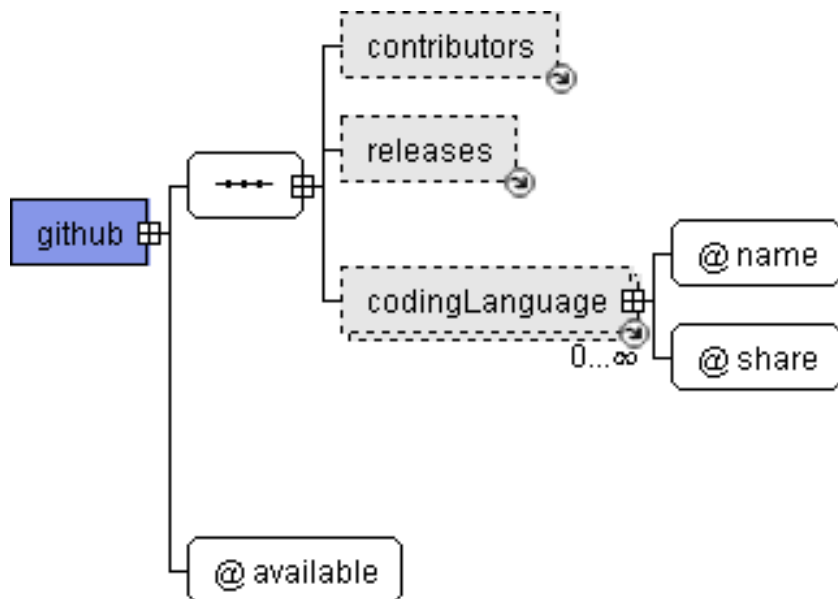
tools__schema.xsd

Schema tools__schema.xsd

element

codingLanguage | contributors | description | github | latestRelease | link |
longDescr | name | releaseMonth | releaseYear | releases | shortDescr | tool |
tools | version

Element github



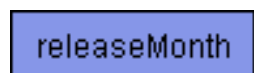
Elements

Element	Type	Documentation
contributors	xs:string	
releases	xs:string	
codingLanguage	Local type	

Attributes

Attribute	Type	Documentation
available	xs:string	No documentation

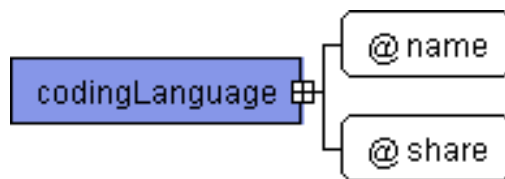
Element releaseMonth



Element link



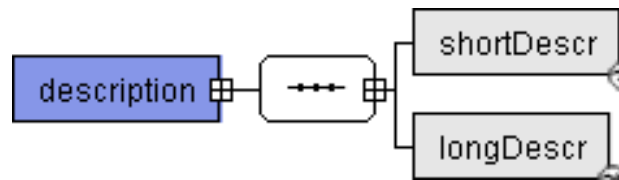
Element codingLanguage



Attributes

Attribute	Type	Documentation
name	xs:string	No documentation
share	xs:string	No documentation

Element description



Elements

Element

Type