Dokumentation zum XML-Projekt im Master-Modul Internettechnologie

Lennart Mende
Richard Mende

HTWK Leipzig

Wintersemester 2025/26

Prof. Dr.-Ing. Andreas Pretschner

2. Januar 2026

Inhaltsverzeichnis

Abkürzungsverzeichnis

Abkürzung	Bedeutung
FO	Formatting Objects
HTML	HyperText Markup Language
PDF	Portable Document Format
XML	eXtensible Markup Language
XSD	XML Schema Definition
XSL	eXtensible Stylesheet Language
XSLT	XSL Transformations

1 Projekübersicht

1.1 Zielstellung

Als Grundlage dieses Projekts dient das Anlegen eines XML-Datensatzes, der aus mehreren Objekten besteht. Jedes Objekt soll mindestens ein Attribut sowie eine Datensequenz enthalten. Auf Basis dieses Datensatzes wird eine XSD-Datei erstellt, sodass die XML-Datei validiert werden kann. Dabei wird überprüft, ob die XML-Datei der definierten Struktur entspricht und alle erforderlichen Angaben enthalten sind. Zusätzlich wird eine XSD-Datei mithilfe der in *EditiX* integrierten automatischen Dokumenterstellung generiert.

Der zweite Teil des Projekts befasst sich mit der Transformation der XML-Datei in zwei unterschiedliche Ausgabeformate. Zunächst sollen die Daten aus der XML-Datei in Form einer HTML-Seite dargestellt werden, wofür eine XSLT-Transformationsdatei erstellt werden muss. Darüber hinaus wird eine FOP-Transformation, um aus der XML-Datei eine PDF-Datei zu generieren. Eine schematische Übersicht dieser Transformationen zeigt Abbildung 1.

Der dritte Aufgabenteil beschäftigt sich mit der XML-Programmierung. Dabei soll eine Applikation erstellt werden, welche die Validierung, Abfrage und Transformation der XML-Datei ermöglicht.

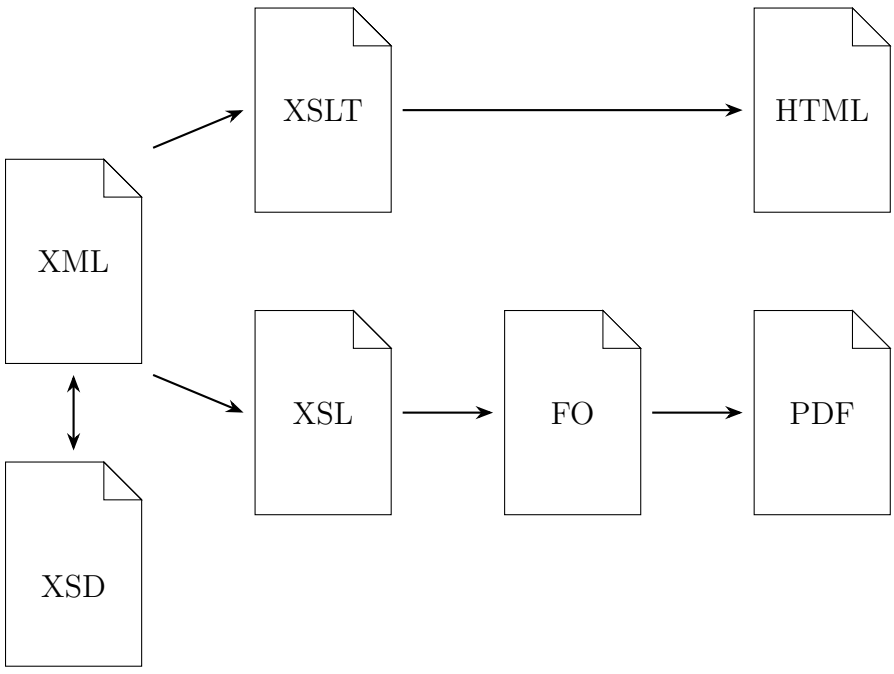

Abbildung 1: Übersicht der Dokumenttypen und Transformationspfade

1.2 Projektorganisation

Zur strukturierten Umsetzung des Projekts sowie zur Sicherstellung einer nachvollziehbaren Versionsverwaltung wurde ein öffentlich zugängliches *GitHub-Repository* unter `https://github.com/rimen13/Projekt_Internettechnologie` eingerichtet. Die Gliederung der Projektdateien ergibt sich folgendermaßen:

```
project/
|-- assets/
|   |-- src/
|   |   |-- products.xml
|   |   |-- products.xsd
|   |   |-- products-automatic-generated.xsd
|   |   |-- products-to-html.xslt
|   |   |-- products-to-pdf.xsl
|   |   |-- products-validation.xml
|   |-- output/
|       |-- products.html
|       |-- products.fo
|       |-- products.pdf
|-- documentation/
|   |-- documentation.pdf
|-- programming/
    |-- src/
    |   |-- query_backend.py
    |   |-- query_from_project.py
    |   |-- query_interactive.py
    |   |-- validate_backend.py
    |   |-- validate_from_project.py
    |   |-- validate_interactive.py
    |   |-- transform_backend.py
    |   |-- transform_from_project.py
    |   |-- transform_interactive.py
    |-- output/
    |   |-- products.html
    |-- requirements.txt
```

Als Entwicklungswerkzeuge kamen der XML-Editor *EditiX* zur Erstellung der Dateien im Ordner `assets` sowie *Visual Studio Code* als integrierte Entwicklungsumgebung zur Umsetzung der XML-Programmierung zum Einsatz. Die inhaltliche Ausarbeitung orientiert sich an den Modul-Unterlagen auf Moodle. Die in der XML-Datei verwendeten Preise und Stückzahlen der elektronischen Bauteile basieren auf den Angaben der Websites `https://www.digikey.de` und `https://www.mouser.de`. Außerdem wurde ChatGPT als Unterstützung bei der sprachlichen Gestaltung der Dokumentation, der FO-Syntax und der Programmierung eingesetzt. Als fachliche Grundlage diente das Buch „Beginning XML“ von David Hunter et al. (2007, 4. Auflage).

2 Datensatz

2.1 XML-Datei

Als Beispieldatensatz für dieses Projekt wurde ein Katalog elektronischer Bauteile eines fiktiven Elektronikshops modelliert. Der vollständige Datensatz ist in der Datei `products.xml` abgelegt und umfasst insgesamt 9 unterschiedliche Bauteile (3 Widerstände, 2 Spulen, 2 Kondensatoren und 2 Dioden). Das XML-Dokument besitzt das Root-Element *products*, unter dem alle *product*-Elemente zusammengefasst sind. Jedes *product*-Element repräsentiert ein einzelnes Bauteil und stellt damit ein Objekt des Datensatzes dar, welches über das Attribut *id* eindeutig zu identifizieren ist. Zusätzlich wird über das Attribut *type* der grundlegende Bauteiltyp spezifiziert, bei Dioden erweitert durch das Attribut *subtype*. Unabhängig vom konkreten Bauteiltyp enthalten alle *product*-Elemente eine einheitliche Menge an Kindelementen zur Beschreibung allgemeiner Produkteigenschaften. Diese sind die Anzahl (*amount*), der Preis (*price*) und der Hersteller (*manufacturer*). Über das *unit*-Attribut lässt sich der Preis mit unterschiedlichen Währungseinheiten angeben.

Ergänzend zu diesen allgemeinen Angaben besitzt jedes Produkt ein oder zwei bauteilspezifische Datensequenzen, welche die relevanten physikalischen Kenngrößen modellieren. Die Bezeichnung dieser Sequenzen ist vom jeweiligen Bauteiltyp abhängig (*resistance* für Widerstände, *inductance* für Spulen oder *capacitance* für Kondensatoren). Für Dioden können mehrere Kenngrößen angegeben werden (*forwardVoltage* und *reverseVoltage*). Diese Datensequenzen enthalten jeweils eine strukturierte Beschreibung des Zahlenwertes, bestehend aus Mantisse (*value*), Exponent (*exponent*), Einheit (*unit*) sowie optional einer Toleranzangabe (*tolerance*). Diese Toleranzangabe erfolgt über ein *unit*-Attribut und kann entweder relativ (in Prozent) oder absolut angegeben werden. Diese explizite Zuordnung der physikalischen Größen zu den jeweiligen Bauteiltypen erleichtert die in Kapitel 2.2 beschriebene Validierung der XML-Datei mithilfe einer XSD-Datei.

2.2 XSD-Datei

Die manuell erstellte XSD-Datei `products.xsd` legt fest, welche Elemente, Attribute und Datentypen in der zu validierenden XML-Datei zulässig sind. Sie bildet damit die Grundlage für eine automatisierte Validierung der XML-Datei und stellt sicher,

dass die definierten strukturellen und typbezogenen Anforderungen eingehalten werden. Zur direkten Validierung der XML-Datei gegen diese XSD-Datei in *EditiX* dient die XML-Datei `products-validation.xml`. Diese enthält einen Verweis auf die XSD-Datei *products.xsd*, sodass die XML-Datei automatisch gegen die XSD-Datei validiert wird.

Die Anzahl der *product*-Elemente muss mindestens 1 betragen. Mithilfe von *choice* werden die zulässigen Kombinationen der physikalischen Größen *resistance*, *inductance*, *capacitance*, *forwardVoltage* und *reverseVoltage* zum jeweiligen Bauteil überprüft.

Alle Produkte besitzen die gemeinsamen Attribute *id*, *type* und optional *subtype*, die in einer *attributeGroup* definiert sind. Die bauteilspezifischen Elemente werden jeweils durch einen eigenen *complexType* beschrieben. Die Typisierung einfacher Elemente ohne Attribute erfolgt entweder direkt über die Schema-Datentypen wie *float*, *short* oder *string*, oder über benutzerdefinierte *simpleTypes* bzw. *complexTypes* mit einfachem Inhalt, wie beispielsweise *ExponentType*, *ToleranceType* oder *PriceType*. Für Attribute werden entweder Standardwerte definiert (z. B. % bzw. EUR für *unit* als Einheit der Toleranz bzw. Währung) oder keine Vorgaben gemacht (z. B. für *id*, *type* und *subtype*). Standardwerte sorgen für konsistente Voreinstellungen bei fehlenden Angaben, während feste Werte Abweichungen erkennen und als Fehler kennzeichnen.

Der Exponent ist auf eine diskrete Menge vordefinierter Werte beschränkt, die ganzzahlige Vielfache von 3 sind und im Bereich von -12 bis 9 liegen. Diese Einschränkung wurde mit *ExponentType* als *enumeration* umgesetzt, da ausschließlich diese Exponenten in den anschließenden Transformationen in Kapitel 3 in physikalische Vorsilben wie µ, m oder k überführt werden können.

Die Datei `products-automatic-generated.xsd` wurde durch die in *EditiX* enthaltene automatische Dokumenterstellung generiert. Sie ist allerdings weniger präzise als die bereits beschriebene XSD-Datei. Dies kann damit begründet werden, dass bei der automatischen Generierung nicht die gleichen präzisen Typdefinitionen vorgenommen werden können. Außerdem ist in dieser XSD-Datei kein *choice* enthalten und daher für die Validierung der XML-Datei nur eingeschränkt geeignet. Es wird daher empfohlen, zur Validierung der XML-Datei die XSD-Datei `products.xsd` zu verwenden. Insbesondere für die in Kapitel 3 beschriebenen Transformationen muss die XML-Datei die Anforderungen der XSD-Datei `products.xsd` erfüllen. Die Anforderungen aus der XSD-Datei `products-automatic-generated.xsd` sind nicht ausreichend.

3 Transformationen von XML-Daten

3.1 XSLT-Transformation von XML zu HTML

Um die Daten aus der XML-Datei auf einer HTML-Website darstellen zu können, wurde eine Transformation mit der erstellten Datei `products-to-html.xslt` durchgeführt. Auf der resultierenden HTML-Website, die unter `products.html` abgelegt ist, wird für alle 4 Bauteilarten jeweils eine eigene Tabelle erstellt, in der die zugehörigen Bauteildaten angezeigt werden. Diese Tabellen sind zunächst zugeklappt und lassen sich durch Anklicken des Dreieck-Symbols (▶ → ▼) aufklappen.

Für die Ausgabe der Tabellen wird zwischen Widerständen, Spulen und Kapazitäten sowie Dioden unterschieden, da sich die Anzahl der angegebenen physikalischen Größen unterscheidet. Die Ausgabe der Bauteildaten erfolgt anhand eines Templates, welches für alle physikalischen Größen identisch ist. Dabei werden über *XPath*-Ausdrücke die relevanten Informationen aus der XML-Datei extrahiert. Jede physikalische Größe enthält Mantisse, Exponent, Einheit und optional Toleranz. Die Exponenten werden dabei als physikalische Vorsilben interpretiert, z. B. 10^{-6} als µ. Mit einer `xsl:when`-Abfrage wird überprüft, ob der optionale Wert für die Toleranz vorhanden ist. Ist dies nicht der Fall, wird ein Bindestrich an der entsprechenden Stelle in der Tabelle angezeigt. Alle weiteren Angaben in der Tabelle sind, mit Ausnahme des Exponenten, obligatorisch und werden stets ausgegeben. Der Exponent ist optional und stellt lediglich einen Bestandteil der physikalischen Größe dar, ohne eine eigene Tabellenzelle zu belegen. Dementsprechend ist auch ohne Angabe eines Exponenten eine korrekte Darstellung der Tabelle gewährleistet.

3.2 FOP-Transformation von XML zu PDF

Die PDF-Ausgabe der Produktdaten erfolgt über eine FOP-Transformation. Die XSLT-Datei `products-to-html.xslt` wurde wiederverwendet, um die relevanten Informationen aus der XML-Datei mittels *XPath* zu extrahieren. Die XSL-Transformation erfolgt anhand der XSL-Datei `products-to-pdf.xsl`, in der im Vergleich zur XSLT-Datei die HTML-Syntax durch FO-Syntax ersetzt und Layout-Anpassungen an der Darstellung vorgenommen wurden. Die Erstellung der FO-Datei `products.fo` und die FOP-Transformation erfolgt in *EditiX* automatisch, sodass anschließend die PDF-Datei `products.pdf` generiert wird.

Ein spezieller Anpassungspunkt besteht in der Darstellung des Symbols Ω als Einheit des Widerstands. Im Vergleich zu anderen Zeichen musste es um 0.145 em nach oben verschoben werden, damit die Zeilenhöhe im Vergleich zu den vorherigen Zeichen erhalten bleibt.

Die in den Kapiteln 3.1 und 3.2 beschriebenen Transformationen sind generisch und können somit für jede XML-Datei durchgeführt werden, welche die Anforderungen der in Kapitel 2.2 erläuterten XSD-Datei erfüllen.

4 Programmierung

4.1 Grundlegende Informationen

Durch die XML-Programmierung sollen folgende Funktionalitäten implementiert werden:

1. Validierung eines XML-Dokuments gegen ein XSD-Schema

2. Abfrage von Informationen aus einem XML-Dokument

3. Transformation eines XML-Dokuments in eine HTML-Website

Zur Umsetzung dieser Funktionalitäten wird im Rahmen dieses Projekts die Programmiersprache *Python* verwendet. Um das die in den folgenden Kapiteln beschriebenen Befehle in der Kommandozeile ausführen zu können, muss *Python* von der offiziellen Website https://www.python.org unter dem Reiter *Downloads* installiert werden. Für dieses Projekt wurde Version 3.12.8 genutzt. Außerdem müssen die beiden Module *lxml* und *tkinter* installiert werden.

Durch das Modul *lxml* ist es möglich, diese Funktionalitäten umzusetzen.

Damit die Befehle über die Kommandozeile aufgerufen werden können, muss zunächst ein lokaler Python Download erfolgen.

HIER MUSS ICH ES MAL MIT MEINEM EIGENEN LINUX SYSTEM PROBIEREN!

Für alle 3 Teilaufgaben existieren 3 getrennte Python-Dateien:

1. Backend-Datei (`*_backend.py`): enthält ausschließlich die Logik der jeweiligen Aufgabe und können nicht direkt ausgeführt werden

2. Projektmodus-Datei (`*_from_project.py`): führt die jeweilige Aufgabe mit den im Projekt erarbeiteten Dateien aus

3. Interaktiv-Datei (`*_interactive.py`): ermöglicht die Auswahl beliebiger Dateien und führt die jeweilige Aufgabe für diese durch

Die jeweiligen Dateien enthalten den beschriebenen Code, der ausführlich kommentiert ist. Die Dateien können ausgeführt werden, indem man zunächst die virtuelle Umgebung aktiviert und in der Kommandizeile in das Verzeichnis `programming/src` wechselt und den Befehl `python desired_file.py` ausführt. Dabei ist `desired_file.py` ein Platzhalter für den eigentlichen Dateinamen, der an dieser Stelle natürlich eingesetzt werden muss.

4.2 Abfrage

Bei der Abfrage werden gezielt Informationen aus einem XML-Dokument extrahiert, indem XPath verwendet wird. Hierbei werden 3 verschiedene Abfragen durchgeführt:

1. Welche Produkt-IDs gibt es?

2. Wie heißen die Hersteller? (Hierbei sollen mehrfach aufgeführte Hersteller nur einmal aufgeführt werden.)

3. Was ist die Gesamtanzahl aller Widerstände im Lager?

Führt man die Datei `query_from_project.py` für den Projektmodus aus, dann wird in der Kommandozeile folgendes ausgegeben:

```
Query 1: Retrieve all product IDs.
Result: ['R001', 'R002', 'R003', 'L001', 'L002', 'C001', 'C002',
        'D001', 'D002']
```

```
Query 2: Retrieve unique manufacturer names.
Result: ['Diodes Incorporated', 'Micro Commercial Components',
        'Murata Power Solutions', 'Samsung Electro-Mechanics',
        'Stackpole Electronics', 'Taiyo Yuden']

Query 3: Calculate the total number of resistors in stock.
Result: 864907
```

Möchte man die interaktive Abfrage durchführen, kann man über die Datei `query_from_project.py` eine XML-Datei aus dem Datei-Explorer auswählen. Anschließend wird zunächst überprüft, ob die angegebene Datei das vorgegebene XSD-Schema erfüllt. Ist dies nicht der Fall, können die Abfragen nicht durchgeführt werden und es wird eine Fehlermeldung ausgegeben. Ist die XML-Datei gültig, werden ebenfalls die 3 Abfragen getätigt und die Ergebnisse ausgegeben.

4.3 Validierung

Um eine XML-Datei gegen ein XSD-Schema validieren zu können, kommen die Dateien `validate_from_project.py` und `validate_interactive.py` zur Anwendung. Dabei kann in ersterem Fall die XML-Datei gegen die XSD-Datei aus dem gegebenen Projekt validiert werden. Für die zweite Datei ist die Auswahl einer beliebigen XML- und XSD-Datei im Datei-Explorer möglich. Die Ausgabe in der Kommandozeile für eine gültige oder nicht gültige XML-Datei lautet folgendermaßen (kommt auch mit automatisch generierter XSD raus):

```
VALIDATION RESULT: products.xml is valid according to products.xsd

VALIDATION RESULT: products.xml is NOT valid according to products.xsd
```

4.4 Transformation

Die Transformation erfolgt über die Dateien `transfrom_from_project.py` und `transform_interact`
Dabei wird entweder die XML- und XSLT-Datei des gegebenen Projekts verwendet

oder im interaktiven Modus werden diese beiden Dateitypen vom Anwender übergeben. Anschließend wird die Transformation in eine HTML-Datei durchgeführt und in den *output*-Ordner (in *programming*) gelegt, sodass sich der Anwender die entstehende HTML-Website anschauen kann. Im interaktiven Modus kann sich der Anwender den Zielordner selsbt aussuchen. In der Kommandozeile kommt der Hinweis:

```
HTML generated: products.html
```

Im Unterordner *programming/output* befindet sich bereits die HTML-Datei, die durch die durchgeführte Transformation für die im Projekt gegebenen Dokumente entstanden ist.