Dokumentation zum XML-Projekt im Master-Modul Internettechnologie

Lennart Mende
Richard Mende

HTWK Leipzig

Wintersemester 2025/26

Prof. Dr.-Ing. Andreas Pretschner

4. Januar 2026

Inhaltsverzeichnis

1 Projektübersicht **1**
1.1 Zielstellung 1
1.2 Projektorganisation 2

2 Datensatz **3**
2.1 XML-Datei 3
2.2 XSD-Datei 3

3 Transformationen von XML-Daten **5**
3.1 XSLT-Transformation von XML zu HTML 5
3.2 FOP-Transformation von XML zu PDF 5

4 Programmierung **6**
4.1 Grundlegende Informationen 6
4.2 Validierung 7
4.3 Abfrage 8
4.4 Transformation 9

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 Projektübersicht

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.

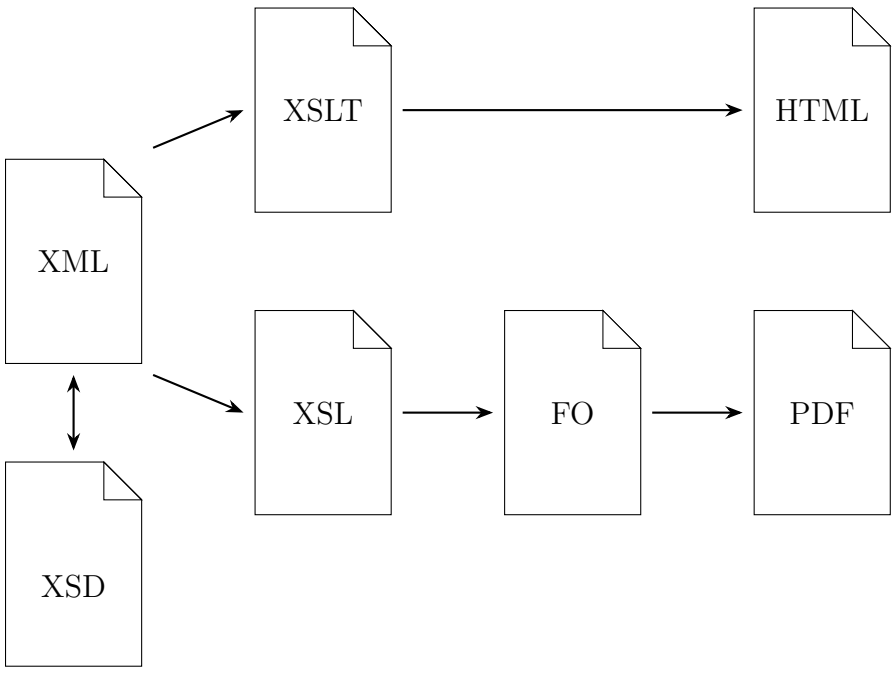

Abbildung 1: Übersicht der Dokumenttypen und Transformationspfade

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

In diesem Projekt sollen folgende Funktionalitäten umgesetzt werden:

1. Validierung einer XML-Datei gegen eine XSD-Datei
2. Abfrage ausgewählter Informationen aus einer XML-Datei
3. XSLT-Transformation einer XML-Datei zu einer HTML-Website

Zur Implementierung dieser Funktionalitäten wurde die Programmiersprache *Python* verwendet, da sie leistungsfähige Bibliotheken für die Verarbeitung von XML-Dateien bereitstellt. Über die Bibliothek `lxml` können XML-Parsing, Schema-Validierung und XSLT-Transformationen durchgeführt werden. Für die Programmierung der Funktionalitäten wurden insgesamt 9 Dateien erstellt, die sich folgendermaßen gliedern:

1. **Backend-Dateien (`*_backend.py`)**
   Diese Dateien enthalten ausschließlich die Logik der jeweiligen Aufgabe (Abfrage, Validierung, Transformation). Sie werden von den Projekt- und Interaktivmodus-Dateien aufgerufen, wobei die entsprechenden Dateipfade übergeben werden.

2. **Projektmodus-Dateien (`*_from_project.py`)**
   Diese Dateien führen die jeweilige Aufgabe für die in diesem Projekt verwendeten Dateien aus. Darin werden die Dateipfade bestimmt und anschließend an die jeweilige Backend-Datei übergeben.

3. **Interaktivmodus-Dateien (`*_interactive.py`)**
   Diese Dateien ermöglichen dem Anwender die Auswahl beliebiger XML-, XSD- oder XSLT-Dateien über ein sich öffnendes Fenster. Dies wird durch die Verwendung des Moduls `tkinter` realisiert. Anschließend werden die Dateipfade an die jeweilige Backend-Datei übergeben.

Um die Dateien auszuführen, muss Python in der Version 3.12.8 (oder neuer) installiert sein. Für dieses Projekt wurden außerdem die Module `lxml` und `tkinter` verwendet, welche über eine virtuelle Umgebung installiert werden können. Der Aufruf der einzelnen Dateien erfolgt über die Kommandozeile. Dafür muss zunächst die virtuelle Umgebung aktiviert und anschließend in das Verzeichnis `programming/src` gewechselt werden. Anschließend kann der Befehl `python validate_from_project.py` ausgeführt werden. Dabei kann `validate_from_project.py` durch die 5 weiteren Dateinamen `validate_interactive.py`, `query_from_project.py`, `query_interactive.py`, `transform_from_project.py` und `transform_interactive.py` ersetzt werden. Die 3 Backend-Dateien können prinzipiell ebenfalls aufgerufen werden, liefern allerdings kein Ergebnis.

4.2 Validierung

Die Validierung eines XML-Dokuments gegen ein XSD-Schema erfolgt in der Datei `validate_backend.py`. Dabei werden sowohl das XML-Dokument als auch die XSD-Datei mithilfe von `etree.parse()` eingelesen. Anschließend wird aus der XSD-Datei ein `XMLSchema`-Objekt erzeugt, welches die in der Schema-Datei definierten Struktur- und Typregeln bereitstellt. Die eigentliche Validierung erfolgt durch den Aufruf der Methode `schema.validate(xml_doc)`. Bei erfolgreicher Validierung wird in der Kommandozeile eine entsprechende Erfolgsmeldung ausgegeben. Schlägt die Validierung fehl, wird zusätzlich das Fehlerprotokoll des Schemas ausgegeben, welches detaillierte Informationen zur Art und Position der Regelverletzungen enthält.

Beim Ausführen der Datei `validate_from_project.py` werden die in diesem Projekt enthaltene XML-Datei und die zugehörige XSD-Datei an die Backend-Datei übergeben. Nach Abschluss der Validierung wird in der Kommandozeile das Ergebnis `VALIDATION RESULT: products.xml is valid according to products.xsd` ausgegeben. Im interaktiven Modus über die Datei `validate_interactive.py` kann der Anwender in einem sich öffnenden Fenster beliebige XML- und XSD-Dateien auswählen.

4.3 Abfrage

Die Abfrage von Daten aus einer XML-Datei ist in der Datei `query_backend.py` implementiert und nutzt *XPath*-Ausdrücke zur gezielten Extraktion von Informationen aus der XML-Datei. Zu Beginn wird das XML-Dokument mithilfe der Funktion `etree.parse(xml_path)` eingelesen. Diese Funktion liest die übergebene XML-Datei ein und erzeugt einen internen XML-Baum, auf dem anschließend *XPath*-Abfragen ausgeführt werden können.

Da das verwendete XML-Dokument einen Namespace definiert, ist es erforderlich, ein entsprechendes Namespace-Mapping zu definieren. Die *XPath*-Abfragen müssen diesen Namespace explizit berücksichtigen, da andernfalls keine Elemente gefunden würden. Anschließend werden drei *XPath*-Abfragen durchgeführt:

1. **Ermittlung aller Produkt-IDs**
   Über den *XPath*-Ausdruck `//p:product/@id` werden alle `id`-Attribute sämtlicher Produkte extrahiert. Das Ergebnis ist eine Liste von Zeichenketten.

2. **Abfrage aller eindeutigen Hersteller**
   Mit dem *XPath*-Ausdruck `//p:product/p:manufacturer/text()` werden alle in der XML-Datei enthaltenen Herstellernamen ausgelesen. Da einzelne Hersteller mehrfach auftreten können, werden mehrfach aufgeführte Hersteller mithilfe der Python-Funktionen `set()` und `sorted()` herausfiltert. Als Ergebnis wird die alphabetisch sortierte Liste der Hersteller ausgegeben.

3. **Berechnung der Gesamtanzahl von Widerständen** Mithilfe eines *XPath*-Ausdrucks werden gezielt Produkte mit dem Attribut `type='resistor'` ausgewählt und deren `amount`-Werte extrahiert. Diese werden anschließend aufsummiert und das Ergebnis ausgegeben.

Die Datei `query_from_project.py` ruft diese Backend-Funktion mit der projektspezifischen XML-Datei auf und gibt die Ergebnisse in der Kommandozeile aus:

```
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
```

Im interaktiven Modus (`query_interactive.py`) kann der Anwender aus einem sich öffnenden Fenster die XML-Datei auswählen, aus welcher er die Daten abfragen möchte. Allerdings können die beschriebenen Abfragen nur für XML-Dateien durchgeführt werden, welche die Validierung gegen die XSD-Datei `products.xsd` erfüllen. Demzufolge wird zunächst über die Backend-Datei der Validierung `validate_backend.py` überprüft, ob die ausgewählte XML-Datei valide ist. Ist dies der Fall, werden auch hier die Ergebnisse der Abfragen in der Kommandozeile ausgegeben. Andernfalls bricht das Programm mit einem entsprechenden Hinweis ab.

4.4 Transformation

Die Transformation einer XML-Datei in eine HTML-Website mithilfe einer XSLT-Datei ist in der Datei `transform_backend.py` implementiert. Zu Beginn werden sowohl die übergebene XML-Datei als auch die zugehörige XSLT-Datei mithilfe der Funktion `etree.parse()` eingelesen. Anschließend wird aus der XSLT-Datei mittels `etree.XSLT()` ein Transformationsobjekt erzeugt. Dieses Objekt enthält die in der XSLT-Datei definierten Template-Regeln. Die eigentliche Transformation erfolgt durch den Aufruf dieses Transformationsobjekts auf das geparste XML-Dokument. Das Ergebnis ist ein neues Dokumentenobjekt, das die transformierte HTML-Struktur enthält und anschließend als HTML-Datei gespeichert wird.

Die Datei `transform_from_project.py` ruft diese Backend-Funktion mit den projektspezifischen XML- und XSLT-Dateien auf und legt die erzeugte HTML-Datei im vorgesehenen Ausgabeverzeichnis ab. Im interaktiven Modus (`transform_interactive.py`) kann der Anwender sowohl die Eingabedateien als auch den Zielpfad frei wählen.

Nach erfolgreicher Transformation wird der Anwender in der Kommandozeile über die Erstellung der HTML-Datei mit dem Hinweis `HTML generated: products.html` informiert. Diese durch die Projektdokumente erzeugte HTML-Datei `products.html` befindet sich bereits im Unterordner `programming/output`.