Bachelorarbeit

Qualitätssicherung von Buchmetadaten in ONIX for Books mit Schematron

Verfasserin: Herta Albrecht

Matrikelnummer: 26200

Studiengang: Technische Redaktion und E-Learning-Systeme

Fachbereich: Ingenieur- und Naturwissenschaften

Hochschule Merseburg

Diese Abschlussarbeit dient zur Erlangung des akademischen Grades

Bachelor of Engineering

Erstprüfer: Dr. rer. nat. Thomas Meinike

Zweitprüfer: Dipl.-Phys. Gerrit Imsieke

Leipzig, 19.02.2023

Kurzzusammenfassung

Diese Bachelorarbeit beschäftigt sich mit der Entwicklung eines Schematron-Schemas für Fehler und Problematiken, die im Buchmetadaten-Austauschformat ONIX for Books 2.1 auftreten können. Die Schematron-Regeln haben dabei den Anspruch, benutzerfreundlich und effizient für potenzielle Nutzer zu sein. Im Zusammenhang mit der Datenqualität in ONIX 2.1 wird darüber hinaus untersucht, ob in der aktuellen ONIX-3.0-Spezifikation signifikante Verbesserungen umgesetzt wurden.

Short summary

This bachelor thesis deals with the development of a Schematron schema for errors and issues that may occur in the book metadata exchange format ONIX for Books 2.1. In doing so, the Schematron rules aim to be user-friendly and efficient for potential users. In the context of data quality in ONIX 2.1, it is furthermore analyzed whether significant improvements have been implemented in the ONIX 3.0 specification.

Vorwort

An dieser Stelle möchte ich mich bei allen bedanken, die mich fachlich und persönlich bei der Verfassung dieser Abschlussarbeit unterstützt haben.

Dabei danke ich besonders Gerrit Imsieke für das interessante, vielseitige Praktikum bei le-tex, das mich zum Thema dieser Arbeit hingeführt hat, und für seine Betreuung.

Ebenso danke ich Dr. Thomas Meinike für die Betreuung während des Erstellens der Arbeit und seine hilfreichen Anregungen.

Besonderer Dank gilt Barbara Herlinger für ihre wertvolle Unterstützung und ihre ausführlichen Informationen in Bezug auf ihre langjährige Erfahrung mit ONIX for Books.

Inhaltsverzeichnis

Abbildungsverzeichnis

Tabellenverzeichnis

Listingverzeichnis

Abkürzungsverzeichnis

CDATA	Character Data
DTD	Document Type Definition
EDItEUR	European Book Sector Electronic Data Interchange Group
HTML	Hypertext Markup Language
ID	Identifier
MVB	Marketing- und Verlagsservice des Buchhandels
ONIX	Online Information Exchange
Regex	Regular Expression
RELAX NG	Regular Language Description for XML New Generation
SQF	Schematron QuickFix
SVRL	Schematron Validation Report Language
TOC	Table of contents
URL	Uniform Resource Locator
VLB	Verzeichnis Lieferbarer Bücher
W3C	World Wide Web Consortium
WWW	World Wide Web
XHTML	Extensible Hypertext Markup Language
XML	Extensible Markup Language
XPath	XML Path Language
XSD	XML Schema Definition
XSL	Extensible Stylesheet Language
XSLT	Extensible Stylesheet Language Transformations

1 Einleitung

Gute Metadaten sind für alle Akteure entlang einer Wertschöpfungskette von wichtiger Bedeutung. Metadaten sind strukturierte Daten mit dem Zweck, Ressourcen jeglicher Art einheitlich zu beschreiben, um damit das Suchen, Finden und Selektieren relevanter Ressourcen aus einer Vielzahl von Informationen zu erleichtern[1]. In der deutschen Buchbranche ist das Metadaten-Austauschformat *ONIX for Books* für bibliografische Daten und Produktinformationen fest etabliert. Fehlerfreie und bestenfalls reichhaltige Informationen zu Buchprodukten fördern deren Vermarktung und Verkauf. Fehlende oder fehlerhafte Informationen, die teilweise zu Verarbeitungsfehlern oder anderen Problematiken führen können, führen hingegen zu Mehraufwand und damit zu höheren Kosten im Arbeitsprozess. Da es heute eine große Zahl an handelbaren Buchprodukten von einer Großzahl von Verlagen gibt, die von unzähligen Buchhändlern – sowohl vor Ort, als auch auf Onlineplattformen – vertrieben werden, ist ein automatisierter und standardisierter Austausch der Informationen unerlässlich für einen effektiven Workflow.

Bei ONIX for Books, im Folgenden kurz „ONIX“ handelt es sich um eine XML-basierte internationale Standardspezifikation für die Kommunikation von Buch-, E-Book- und digitalen Audio-Metadaten. Da es sich um ein Kommunikations-Austauschformat handelt, werden die ONIX-Dateien auch *ONIX-Meldungen* (engl. *ONIX messages*) genannt. Die Akteure, deren Arbeitsprozesse Buchmetadaten beinhalten, sind in erster Linie Verlage, Dienstleister, Distributoren, Händler sowie weitere Zwischenhändler.[2] Da der Metadaten-Austausch zwischen den Akteuren weitgehend automatisiert abläuft, können sich jedoch auch Fehler einschleichen, die zunächst unbemerkt bleiben. Bei einem genaueren Blick auf Websites von Verlagen oder Buchhändlern lassen sich Unregelmäßigkeiten und Fehler bei Angaben zu angebotenen Büchern finden, die potenzielle Leser von einem Kauf abhalten können. Die Fehler können bereits in den Metadaten vorliegen und somit zwischen den Akteuren ausgetauscht worden sein.

Seit der Veröffentlichung der ersten ONIX-Version im Jahr 2000 bis zum Release von ONIX 3.0 im Jahr 2009 haben sich die Anforderungen an den Buchhandel und der Stand der Technik stark verändert. Dennoch ist im deutschen Buchmarkt bis heute die ONIX-Version 2.1 gängig[3], die 2006 veröffentlicht wurde und seit 2014 nicht mehr weiterentwickelt wird. Obwohl die Organisation *EDItEUR* dringend empfiehlt auf ONIX 3.0 zu migrieren, findet die Umstellung aus mehreren Gründen nur nach und nach

[1] Vgl. *Rühle, S.*, Kleines Handbuch - Metadaten, 2012, S. 2.
[2] Vgl. *EDItEUR*, ONIX FAQs.
[3] Vgl. *ebd*.

in der Branche statt.[4] Dass Metadatenmanager nicht die aktuelle ONIX-Version nutzen, hat jedoch zur Folge, dass Kompromisse gemacht werden, wenn beispielsweise ein Element für einen gewünschten Gebrauch in ONIX 2.1 nicht existiert. Dies hat zur Folge, dass ONIX-Elemente anders gebraucht werden, als es die Spezifikation vorsieht: Ein Beispiel dafür ist, dass ein Element, das für den Untertitel vorgesehen ist, einen kurzen Werbetext enthält, da in ONIX 2.1 kein entsprechend signifikantes ONIX-Element existiert.

Der Verlagsdienstleister *le-tex publishing services* ist der Praxispartner dieser Abschlussarbeit, der bereits während eines Studienpraktikums die Idee äußerte, Zweckentfremdungen und einen uneinheitlichen Gebrauch in ONIX-2.1-Elementen zu untersuchen. *Zweckentfremdeter Gebrauch* bedeutet dabei die Nutzung von ONIX-Elementen abweichend von der ONIX-Standardspezifikation. Da allerdings bei den untersuchten ONIX-Dateien zusätzlich andere Problematiken und Fehler auftauchten und auch von Metadatenmanagern berichtet wurden, scheint es sinnvoll, gemeinsam auf diese Zweckentfremdungen, Fehler und Problematiken einzugehen. Weiterhin soll, aufgrund Interessensbekundungen von Geschäftspartnern in Bezug auf eine Migration zu ONIX 3.0, untersucht werden, ob sich hinsichtlich der Problematiken in ONIX 3.0 erkennbare Verbesserungen aufgetan haben. Den Vorschlag, für die Untersuchungen Schematron-Prüfregeln zu erstellen, machte ebenso der Praxispartner. Weil die Schematron-Prüfregeln jedoch auch über die Untersuchung hinaus die Möglichkeit geben, Fehler und Problematiken in ONIX-Dateien aufzudecken, um sie anschließend zu korrigieren, bietet es sich an, mithilfe des Schematron-Schemas die Qualität in den Metadaten zu überprüfen und anschließend optimieren zu können. Um einen Mehrwert aus den Prüfregeln zu erzielen, soll das Schematron-Schema an die Benutzung durch Metadatenmanager angepasst werden und somit benutzerfreundlich sein und Lösungshinweise geben. Geplant ist im Anschluss an diese Abschlussarbeit weiteren ONIX-Nutzern das Schematron-Schema durch den Praxispartner open source[5] zur Verfügung zu stellen.

Da das ONIX-Dateiformat für die Computer-to-computer-Kommunikation entwickelt wurde, ist es für den Menschen eher schwierig zu lesen.[6] EDItEUR liefert auf seiner Website[7] neben älteren und der aktuellen ONIX-Spezifikation eine Vielzahl von Onlinedokumenten, die als Grundlage zum Verstehen und Interpretieren von ONIX-Dateien genutzt werden. Für diese Arbeit werden auch Onlinedokumente von weiteren Organisationen in der Buchbranche und nationalen ONIX-Anwendergruppen in Betracht

[4] Vgl. *EDItEUR*, Frequently Asked Questions about ONIX for Books, 2009, S. 1 f.
[5] https://github.com/transpect/schema-onix
[6] Vgl. *EDItEUR*, ONIX FAQs.
[7] https://www.editeur.org/8/ONIX/

gezogen, da hier oftmals Stärken und Schwächen vom Buchmetadatenaustausch thematisiert und auch Leitfäden herausgegeben werden.

EDItEUR stellt Schemas für ONIX in den Formaten DTD- und XML Schema (XSD)[8] zur Verfügung. Diese Schemasprachen überprüfen, ob ein ONIX-XML-Dokument sich an die im Schema formulierten „Grammatik"-Regeln hält, wie beispielsweise, in welcher Reihenfolge und Anzahl welche Elemente in einem bestimmten Kontext auftreten dürfen. Im Unterschied und als Erweiterung dazu können Schematron-Schemas sog. Business Rules überprüfen. Eine Business Rule ist jede mögliche Bedingung, die an ein XML-Dokument gestellt werden kann, woraufhin diese Bedingungen überprüft werden können.[9] Ein Beispiel für eine Business Rule ist, dass die Schreibweise eines Autornamens im Klappentext genau mit der entsprechenden Namensangabe im Autornamen-Metadatenfeld übereinstimmt. Da Schematron sich gut in XML-Workflows integrieren lässt und die Erstellung individueller nutzerfreundlicher Prüfregeln mit individualisierten Fehlermeldungen ein Vorteil der Schemasprache ist, ist es praktikabel ein Schematron-Schema für ONIX-Dateien zu erstellen. In dieser Arbeit wird der Standard *ISO Schematron*[10] verwendet. Das Schematron-Schema gilt dabei für ONIX-Dateien des deutschsprachigen Raums und es wird bedacht, welche Möglichkeiten genutzt werden können, um die Schematron-Prüfregeln effizient und nutzerfreundlich zu gestalten. Die meisten Prüfregeln lassen sich jedoch für andere Sprachen und Regionen anpassen. Das beinhaltet i. d. R. nicht nur eine Übersetzung der natürlichsprachlichen Meldungstexte, sondern mitunter auch eine Lokalisierung einiger in der Sprache XPath (s. **Abschnitt 2.2.1**) formulierten *XSLT Matching Patterns* (s. **Abschnitt 2.2.2**), also der Muster für die Fehlersuche. Diese Muster enthalten z. B. reguläre Ausdrücke (s. **Abschnitt 2.3**), die auf Zahlwörter in natürlicher Sprache *matchen*.

Als Entwicklungsumgebung wird der weit verbreitete *Oxygen XML Editor* verwendet, der neben einer Schematron-Implementierung auch unterstützende Einstellungen für die Verarbeitung von *XPath* und *XSLT* besitzt. Für genauere Informationen von XPath- und XSLT-Funktionen ist die Online-Referenz „XSLT- und XPath-Funktionen in alphabetischer Reihenfolge"[11] von data2type zu empfehlen. Für ein Schematron-Tutorial sowie eine Referenz ist das Werk „Schematron – Effiziente Business Rules für XML-Dokumente" empfehlenswert. Zur Hilfe der Erstellung und Testung *regulärer Ausdrücke* ist die online verfügbare Software Regexpal[12] ein geeignetes Tool. Zum Verstehen von ONIX-Dateien wurden die von EDItEUR online zur Verfügung gestellten ONIX-Spezifikationen und Codelists genutzt. Zur Untersuchung liegen zwölf ONIX-Dateien von unterschiedlichen

[8] Vgl. *EDItEUR/Bell, G.*, Using local DTD and XSD files after ONIX 2.1 sunset, 2014, S. 1 f.
[9] Vgl. *Hedler, M./Montero Pineda, M./Kutscherauer, N.*, Schematron, 2011, S. 49 f.
[10] Vgl. *Siegel, E.*, Schematron, 2022, S. 9.
[11] https://www.data2type.de/xml-xslt-xslfo/xslt/xslt-und-xpath-referenz/alphabetische-liste
[12] https://www.regexpal.com/

Verlagen vor, bei welchen es teilweise ursprünglich zu Verarbeitungsproblemen und Reklamationen kam. Von diesen zwölf Dateien waren zehn ONIX-2.1- und zwei ONIX-3.0-Meldungen.

An erster Stelle hat die vorliegende Arbeit das Hauptziel, ein Schematron-Schema zu erstellen, das es ermöglicht, die Qualität von Buchmetadaten in einzelnen ONIX-2.1-Dateien effizient zu optimieren. Dazu sind mehrere Teilaufgaben zu bewältigen, wie die Erhebung der Fehler und Problematiken in ONIX-2.1-Dateien, der Klassifizierung der Fehler und Problematiken und ihrer anschließenden Formulierung in Schematron. Die Erstellung eines nutzerfreundlichen Schematron-Schemas für ONIX 2.1, das allerdings auch in ONIX 3.0 anwendbar sein soll, ist ein weiteres, großes Teilziel. Die vorliegende Arbeit dokumentiert den Prozess der Erstellung der Schematron-Prüfregeln auf Grundlage bereits berichteter und selbst entdeckter Problematiken in ONIX-Dateien. Da immer mehr Marktteilnehmer auf ONIX 3.0 umstellen[13], sollen die Optimierungen von ONIX 3.0 in Bezug auf 2.1 nicht außen vor gelassen werden. Die Änderungen von ONIX 3.0 sollen insbesondere unter dem Gesichtspunkt untersucht werden, ob sie helfen, die zuvor genannten Zweckentfremdungen und Problematiken zu vermeiden. Aus diesem Grund wird die Anwendung der Business Rules auf ONIX-3.0-Dateien betrachtet und es werden die Spezifikationen beider Versionen verglichen.

Der Aufbau dieser Abschlussarbeit wird im Folgenden vorgestellt: Zu Beginn werden theoretische Grundlagen vermittelt. Um die Struktur von ONIX-Dateien verstehen zu können, werden die wichtigsten Aspekte der Auszeichnungssprache *XML* erläutert. Für die Erstellung des Schematron-Schemas sind die Sprachen XPath und XSLT sowie reguläre Ausdrücke von Bedeutung. Hier werden auch die für das Schematron-Schema bedeutendsten Elemente, Funktionen und Zeichen kurz erläutert. Danach werden theoretische Grundlagen der Schemasprache DTD, die in ONIX auch Anwendung findet, und der Schemasprache Schematron mit den wichtigsten Elementen vermittelt.

Anschließend wird als Basis für die Signifikanz von Metadaten behandelt, welche Aspekte eine gute Qualität von Metadaten ausmacht, wieso korrekte Metadaten wichtig sind und wie generell Qualität in XML-Dateien gesichert werden kann.

Im vierten Kapitel wird die Historie von ONIX mit seinen Herausforderungen und Möglichkeiten zur Validierung sowie Fehlerbeseitigung behandelt, und es wird der grundlegende Aufbau einer ONIX-Datei beschrieben. Dazu werden auch die Nutzung der ONIX-Spezifikation und ONIX-Codelists zum Interpretieren einer ONIX-Datei erläutert.

Anschließend werden im fünften Kapitel die berichteten und ausfindig gemachten Fehler und Problematiken nach und nach erläutert. Diese werden als Business Rules formuliert und daraufhin sortiert und klassifiziert.

---

[13] Vgl. *Pufe, M.*, Wichtige ONIX-Elemente und Reklamationen, 2022.

Im sechsten Kapitel werden vier Methoden behandelt, wie das Schematron-Schema entsprechend der Zielgruppe nutzerfreundlich gestaltet werden kann, bevor die Erstellung jeder einzelnen Schematron-Prüfregel für das Schema für ONIX 2.1 erläutert wird. Anschließend wird kurz darauf eingegangen, wie ein Fehlerreport unabhängig von der Entwicklungsumgebung erzeugt werden kann.

Im letzten Kapitel vor dem Fazit wird beschrieben, wie die erstellten Schematron-Regeln auch auf die Verwendung in ONIX 3.0 angepasst werden können, und es wird erörtert, wo sich in ONIX 3.0 Veränderungen, insbesondere Verbesserungen, aufgetan haben.

Im Anhang A und B sind Schematron-Schemas mit allen Patterns für ONIX 2.1 sowie das finale Schema für ONIX 2.1 und ONIX 3.0 angegeben. In Anhang C befindet sich eine eigens erstellte ONIX-Beispieldatei, die fünf „absichtlich“ eingefügte Problematiken beherbergt, damit diese ONIX-Datei gegen eines der beiden Schematron-Schemas validiert werden kann, um diese Arbeit nachvollziehbarer zu machen. Dafür ist jeweils der Text herauszukopieren und in eine entsprechende Entwicklungsumgebung einzufügen. In Anhang D ist ein Listing einer XSLT-Pipeline dargestellt, die zur Erzeugung eines Schematron-HTML-Reports dient. In Anhang E befindet sich eine Referenzliste der im Fließtext und Schema verwendeten ONIX-Elemente.

Wenn im Text auf eine Website verwiesen wird, so wird in der Fußnote der Link dazu angegeben. Falls in einer Fußnote oder im Literaturverzeichnis eine Onlinequelle angegeben ist, die zu einem Download führt, wird mit der Bemerkung „führt zu Download“ darauf hingewiesen. Wird im Fließtext auf Stellen in einem Listing oder im Anhang verwiesen, so werden die Stellen mit einer Ziffer in eckigen Klammern bezeichnet.

2 Theoretische Grundlagen

In den kommenden Abschnitten werden die Grundlagen der Technologien erläutert, die für die Erstellung des Schematron-Schemas für ONIX-Dateien notwendig sind. Dazu werden die für diese Arbeit wichtigsten Elemente, Funktionen und Ausdrücke kurz aufgezeigt.

2.1 XML (Extensible Markup Language)

Die *Extensible Markup Language*, kurz XML, ist eine Auszeichnungssprache, die durch ihre einfache Möglichkeit der Strukturierung von Inhalten zu einem globalen Standard wurde. Mit XML ist es möglich, die Struktur, den Inhalt und die Darstellung eines Dokuments zu trennen und nachträglich auch unabhängig voneinander zu ver- oder bearbeiten.

Mithilfe von XML lassen sich spezifische Vokabulare festlegen, um Elemente, aus welchen Dokumente oder andere strukturierte Datenobjekte zusammengesetzt sind, passend zu beschreiben. Daten und Auszeichnungen werden in simpler Textform abgelegt, wofür ein Texteditor ausreicht. Durch die Verschachtelung inhaltlicher Strukturen in beliebiger Tiefe können Hierarchien repräsentiert werden. *Extensible* (engl. *extensible*: erweiterbar) beleuchtet zudem den generischen Charakter der Auszeichnungssprache, weshalb XML auch als eine Inhaltsbeschreibungssprache oder Metasprache bzw. Metaauszeichnungssprache bezeichnet werden kann.[14] Gleichzeitig steht mit XML ein Datenaustauschformat zur Verfügung, das universal eingesetzt werden kann.

Wichtige Anwendungsbereiche von XML sind die Dokumentauszeichnung, das Content-Management wie z. B. die Pflege von Metainformationen und der Daten- bzw. Informationsaustausch.[15]

Es existieren viele Auszeichnungssprachen, die mithilfe von XML definiert wurden. Diese Auszeichnungssprachen werden *XML-Anwendungen* genannt und beinhalten festgeschriebene XML-Vokabulare oder Dokumenttypen, die für bestimmte Bereiche festgelegt wurden. Beispiele für XML-Anwendungen sind *XHTML* (Extensible Hypertext Markup Language), *SVG* (Scalable Vector Graphics) oder *MathML* (Mathematical Markup Language).[16] Auch ONIX for Books ist ein XML-basiertes standardisiertes Dateiformat, das zum Informationsaustausch über Bücher und buchbezogene Produkte zwischen Computersystemen dient.[17]

---

[14] Vgl. *Vonhoegen, H.*, XML, 2018, S. 31f.
[15] *Ebd.*, S. 42–45.
[16] Vgl. *ebd.*, S. 36.
[17] Vgl. *EDItEUR*, ONIX for Books: Product Information Format Specification, 2021, S. 16.

2.1.1 Aufbau

In **Listing 1** ist beispielhaft ein XML-Dokument abgebildet. Jedes XML-Dokument *sollte* mit einem Prolog beginnen, der die sog. XML-Deklaration in der ersten Zeile angibt **[1]**. Optional können im Prolog Verarbeitungsanweisungen (engl. *processing instructions*) für weiterverarbeitende Programme folgen, wie etwa eine Stylesheet-Zuordnung, oder auch Verweise auf bestimmte Schema-Dokumente, wie z. B. auf eine DTD (Document Type Definition). Anschließend folgen die XML-Daten in Form von mit Markierungen ausgezeichnetem Text **[2]**.[18] Das erste Element in einem XML-Dokument nach einer Deklaration bildet das Wurzelelement, welches alle anderen Elemente enthält. In diesem Beispiel ist es das Element mit dem Elementnamen `<dozentenliste>`. So bildet jedes XML-Dokument eine Baumstruktur von Elementen. Elemente beschreiben die Struktur eines XML-Dokuments und enthalten wiederum andere Elemente, Text, beides oder sind leer. Bei leeren Elementen ist es auch erlaubt, eine verkürzte Schreibweise zu verwenden, z. B. `<img/>` für `<img></img>` **[3]**.

```
<?xml version="1.0" encoding="UTF-8"?>                    [1]
  <dozentenliste>                                         [2]
    <dozent anrede="Herr">
      <name>Maier</name>
      <vorname>Fritz</vorname>
      <img src="maier.jpg"/>                              [3]
    </dozent>
    <dozent anrede="Frau">
      <name>Müller</name>
      <vorname>Sabine</vorname>
      <img src="mueller.jpg"/>
    </dozent>
    <!-- Platz für weitere Dozenten -->
  </dozentenliste>
```

Listing 1: Aufbau eines XML-Dokuments (vgl. Becher, M., XML, 2021, S. 15)

Jedes Element wird jeweils durch ein Start-Tag und ein End-Tag begrenzt. **Abbildung 1** zeigt den Aufbau eines Elements in XML. Im End-Tag erscheint vor dem Elementamen ein Slash.[19]

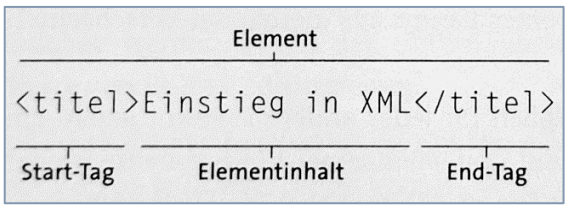

Abbildung 1: Aufbau eines Elements (Vonhoegen, H., XML, 2018, S. 54)

[18] Vgl. *Becher, M.*, XML, 2021, S. 8.
[19] Vgl. *Vonhoegen, H.*, XML, 2018, S. 54 f.

Enthalten Elemente wiederum Elemente, werden sie strukturierte Elemente genannt. Durch die Verschachtelung bilden sich Hierarchien, die zum Beispiel als Eltern- und Kindelemente gekennzeichnet sind. Darüber hinaus können auch noch Kommentare enthalten sein, die mit `<!-- -->` ausgezeichnet sind.

Jedes Element kann beliebig viele *Attribute* enthalten, welche als Name-Wert-Paare angegeben werden, wie im **Listing 1** der Attributname `anrede` mit dem Wert `Herr`. Attribute enthalten in der Regel Zusatzinformationen oder Metadaten über den Inhalt. Innerhalb eines Elementes muss jeder Attributname eindeutig sein. Der gleiche Attributname darf jedoch in verschiedenen Elementen verwendet werden.[20]

2.1.2 Wohlgeformtheit

Mit sogenannten XML-Parsern werden XML-Dokumente gelesen und auf Korrektheit geprüft. Dabei ist die Wohlgeformtheit die Mindestvoraussetzung, damit ein XML-Dokument weiterverarbeitet werden kann. Wohlgeformt heißt ein Dokument, „wenn es den Syntaxregeln und Wohlgeformtheitsbeschränkungen der XML-Spezifikation genügt“[21]. Die Grundregeln für ein wohlgeformtes XML-Dokument lauten:

- Es muss ein Dokumentelement (auch Wurzelelement genannt) enthalten.
- Im Dokumentelement müssen alle anderen Elemente eingeschlossen sein.
- Die Elemente müssen korrekt verschachtelt sein, d. h. in der umgekehrten Reihenfolge beendet werden, in der sie geöffnet werden.
- Für jedes Start-Tag muss ein passendes End-Tag existieren.[22]

Außerdem gibt es Regeln für die Namensgebung der Elemente. So muss z. B. der Name mit einem Buchstaben oder Unterstrich beginnen und die Elementnamen sind *case-sensitive*, was bedeutet, dass z. B. `<Name>...</name>` nicht zulässig ist.[23]

2.1.3 Validierung

Grundsätzlich wird bei der Prüfung von XML-Dokumenten zwischen der Prüfung der Wohlgeformtheit und der Gültigkeit unterschieden. So kann ein XML-Dokument syntaktisch korrekt sein, aber auf der inhaltlichen Ebene Fehler beinhalten, die z. B. bei der Weiterverarbeitung durch Anwendungsprogramme weitere Probleme oder Fehler verursachen können.[24]

Die formale Prüfung auf Wohlgeformtheit kann noch um eine Prüfung auf Gültigkeit durch eine DTD oder ein XML-Schema ergänzt werden. Dazu wird das XML-Dokument

---

[20] Vgl. *Becher, M.*, XML, 2021, S. 10–14.
[21] Vgl. *ebd.*, S. 18.
[22] Vgl. *Vonhoegen, H.*, XML, 2018, S. 617.
[23] Vgl. *Becher, M.*, XML, 2021, S. 10 f.
[24] Vgl. *Vonhoegen, H.*, XML, 2018, S. 60.

von einem *validierenden* Parser gelesen.[25] Weiterhin erlaubt Schematron die Überprüfung von Regeln, die nicht durch eben genannte Schemasprachen abgedeckt werden können, aber komplementär zu diesen Sprachen verwendet werden kann.[26]

2.2 XSL (Extensible Stylesheet Language)

Das W3C definiert XSL als „family of recommendations for defining XML document transformation and presentation“[27] und gibt an, dass die Sprachfamilie aus XSLT, XPath und XSL-FO besteht. Bei XSL handelt es sich um eine XML-Anwendung, die eine Art der Verarbeitung von XML-Dokumenten beschreibt. XSL steht für *Extensible Stylesheet Language* und lässt sich in zwei Untergruppen unterteilen:

1. XSL-FO (XSL Formatting Objects), das die Druck-Präsentation zum Ziel hat.
2. XSLT (XSL Transformations), das auf die Umformung von XML-Dokumenten ausgelegt ist. Damit eng verbunden ist die Pfadbeschreibungssprache XPath, die in XSLT eingebettet der Steuerung und der unmittelbaren Erzeugung von Inhalten dient.

In dieser Arbeit sind XSLT und XPath von größerer Bedeutung. Es existieren bereits mehrere Versionen beider Sprachen, wobei bei der Verarbeitung von XML-Dokumenten die Versionen zusammenpassen müssen; so wird im Zusammenhang mit XSLT 2.0 auch XPath 2.0 verwendet.[28] Im Jahr 2007 wurden XPath 2.0 und XSLT 2.0 zeitgleich als Empfehlung vom W3C verabschiedet.[29]

2.2.1 XPath (XML Path Language)

XPath ist eine vom W3C empfohlene Navigationssprache für XML-Dokumente. Dabei ist sie keine kompilierbare oder ausführbare Programmiersprache, sondern benötigt immer eine andere Sprache, die sie einbettet bzw. ein Tool, das diese Sprache interpretieren kann. Umgekehrt benötigen auch bestimmte Sprachen wie z. B. XSLT und Schematron XPath-Ausdrücke, um funktionieren zu können. Wie **Abbildung 2** verbildlicht, stellt XPath eine Teil- und Schnittmenge von der Abfragesprache XQuery, der Schemasprache XML Schema (XSD), XSLT und auch Schematron dar.[30]

---

[25] Vgl. *Vonhoegen, H.*, XML, 2018, S. 25 f.
[26] Vgl. *Hedler, M./Montero Pineda, M./Kutscherauer, N.*, Schematron, 2011, S. 19.
[27] *W3C*, The Extensible Stylesheet Language Family (XSL), 2017.
[28] Vgl. *Bongers, F.*, XSLT 2.0 & XPath 2.0, 2008, S. 28.
[29] Vgl. *Becher, M.*, XML, 2021, S. 184.
[30] Vgl. *Saxonica*, Technology, 2022.

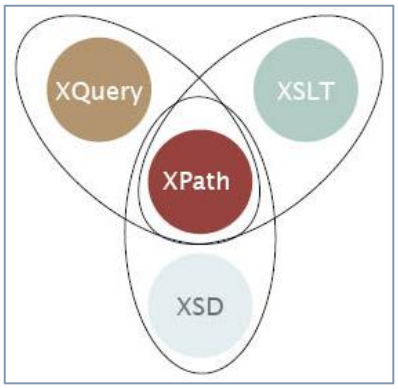

Abbildung 2: XPath-Schnittmenge (Saxonica, Technology, 2022)

Die logische Struktur eines XML-Dokuments spiegelt sich in einer hierarchischen Baumstruktur wider, in der die unterschiedlichen Informationseinheiten in sog. *Knoten* erscheinen. Ein zu verarbeitendes XML-Dokument wird zunächst durch den XML-Parser in eine entsprechende Form gebracht, die als *XPath Data Model* bezeichnet wird.

Ein XPath-Ausdruck kann sehr einfach, aber auch sehr komplex aufgebaut sein, wobei sich komplexe Ausdrücke aus mehreren einfachen Ausdrücken zusammensetzen, die mittels vordefinierter Schlüsselwörter, Symbolen und Operatoren verknüpft werden. Die Pfadausdrücke können Informationen entsprechend bezeichneter Element-, Textknoten, ganzer Zweige und weiterer Teile extrahieren, filtern oder anderweitig verarbeiten. Für vielfältige Verarbeitungsmöglichkeiten sowie die Generierung auszugebender Daten existiert auch eine Reihe von Funktionen, die in XPath-Ausdrücke eingebettet werden können.[31]

Eine Instanz der XPath-Sprache, also jede XPath-Beschreibung bzw. jede gültige Suchanfrage, wird *Ausdruck* genannt und wird ausgewertet, um ein Objekt zu erhalten, für das eine umfangreiche Hierarche an Datentypen möglich ist. Wichtige Datentypen, die schon in XPath 1.0 existierten, sind z. B. eine ungeordnete Datenmenge, die auch leer sein kann, eine Zeichenfolge, die auch leer sein kann, eine Fließkommazahl oder die Werte *true* oder *false*. Bei der Auswertung sind zwischen den verschiedenen Datentypen Umwandlungen möglich.[32] Das Ergebnis der Auswertung eines Ausdrucks ist immer eine Sequenz, also eine Folge von abgegrenzten und unterscheidbaren Einheiten, die als *Items* bezeichnet werden. Dabei ist die Reihenfolge der Items, die in der Sequenz auftreten, relevant. Ein Item kann entweder einen Knoten[33] in einem XML-Dokument repräsentieren oder einen atomaren, also nicht weiter unterteilbaren Wert, darstellen. Beispiele für atomare Werte sind Zahlen oder Zeichenketten. Enthält eine Sequenz kein Item, so gilt sie als leere Sequenz. Diese ist der Rückgabewert eines XPath-Ausdrucks, der kein Ergebnis erzeugt, jedoch *typneutral* ist und somit niemals einen Typfehler

[31] Vgl. *Bongers, F.*, XSLT 2.0 & XPath 2.0, 2008, S. 256–258.
[32] Vgl. *Vonhoegen, H.*, XML, 2018, S. 188.
[33] Beispielsweise Element-, Attribut- oder Textknoten.

verursacht. Es ist somit egal, ob eine Funktion eine leere Sequenz in einem numerischen oder einem String-Kontext zurückgibt, allerdings geben String-Funktionen anstelle einer leeren Sequenz explizit einen leeren String zurück.[34]

In XPath werden verschiedene Arten von Ausdrücken unterschieden:

- Pfadausdrücke (Lokalisierungspfade, Achsen, XPath-Knotentest, XPath-Prädikate)
- Elementare Ausdrücke (Literale, Variablen, Funktionsaufrufe, Klammernausdrücke)
- Arithmetische Ausdrücke (+, -, *, div und weitere)
- Logische Ausdrücke (and, or, not).[35]

Im folgenden Abschnitt werden Pfadausdrücke genauer thematisiert, da diese wichtig für die Nutzung von XPath in Schematron sind.

2.2.1.1 Pfadausdrücke

In „Schematron – Effiziente Business Rules für XML-Dokumente“ beschreiben die Autoren die Navigation durch XPath wie folgt:

> „Ähnlich, wie bei Dateisystemen durch die Pfadnotation ein ‚Navigieren‘ durch den Dateibaum erreicht wird, existiert auch in der XML-Welt eine Syntax, um in XML-Bäumen […] zu navigieren“[36].

Gemeint ist damit die Möglichkeit, zu allen Knoten eines XML-Datenbaumes zu navigieren, während auch die Beziehungen zwischen den einzelnen Knoten eine wichtige Rolle spielen. In XPath werden die Knoten in verschiedene Knotentypen unterteilt, wobei für das zu erstellende Schematron-Schema vor allem Elementknoten von Bedeutung sind. Der Dokumentknoten, auch Wurzelknoten genannt, ist namenlos und sein Inhalt umfasst das ganze Dokument.[37] Durch Elementknoten wird jedes Element des Dokuments repräsentiert; dabei kann ein Elementknoten auch weitere als Kinder enthalten. [38]

XPath-Ausdrücke beschreiben Zielknoten innerhalb der Baumstruktur, in dem jeder Knoten eine unverwechselbare Identität besitzt.[39] Dabei befindet sich die Baumwurzel oben, d. h. der oberste Knoten in der Darstellung ist der Dokumentknoten, dessen einziger nach unten gerichteter Ast zum Wurzelelement zeigt. Von dort aus wird der Baum in Dokumentreihenfolge nach unten verfolgt, wobei dies exakt der Reihenfolge

---

[34] Vgl. *Bongers, F.*, XSLT 2.0 & XPath 2.0, 2008, S. 258.
[35] Vgl. *Becher, M.*, XML, 2021, S. 196 f.
[36] *Hedler, M./Montero Pineda, M./Kutscherauer, N.*, Schematron, 2011, S. 34.
[37] Der Dokumentknoten des Datenmodells ist nicht das Wurzelelement des XML-Dokuments; hier handelt es sich um zwei verschiedene Konzepte (vgl. *Bongers, F.*, XSLT 2.0 & XPath 2.0, 2008, S. 256.)
[38] Vgl. *Becher, M.*, XML, 2021, S. 188.
[39] Vgl. *Bongers, F.*, XSLT 2.0 & XPath 2.0, 2008, S. 256.

entspricht, in der die den Knoten entsprechenden Teile innerhalb des XML-Dokuments erscheinen. Dabei findet immer eine Tiefensuche statt; es werden also immer erst die Kinder und Kindeskinder des aktuellen Knotens abgearbeitet.[40] **Abbildung 3** zeigt auf der rechten Seite die stark abstrahierte Struktur eines Dokumentbaums[41], die zur Vereinfachung nur die Elementknoten darstellt mit der Reihenfolge, in der der Prozessor die Knoten durchläuft. Der Lesevorgang beginnt mit dem Start-Tag des Wurzelelements `<eins>` und wird mit dem Start-Tag des Elements `<zwei>` fortgesetzt. In `<zwei>` sind die Elemente `<drei>` und `<vier>` enthalten. Als nächstes kehrt der Lesevorgang zum End-Tag von `<zwei>` zurück und springt zum Start-tag von `<fuenf>`. Nach dem Lesen des in `<fuenf>` enthaltenen Elements `<sechs>` wird schließlich zum End-Tag des Wurzelelements `<eins>` zurückgekehrt, womit der Lesevorgang beendet ist. In jedem Knoten fallen Start- und End-Tag eines Elementes zusammen.

In der Tag-Darstellung auf der linken Seite der Abbildung ist die Dokumentreihenfolge bzw. Leserichtung deutlicher zu erkennen; hier entspricht sie auch der menschlichen Lesegewohnheit mit der Abfolge von öffnenden und schließenden Tags. Wegen der fortlaufenden Abfolge der vorliegenden Inhalte wird diese Darstellung als *serialisiert* bezeichnet.[42]

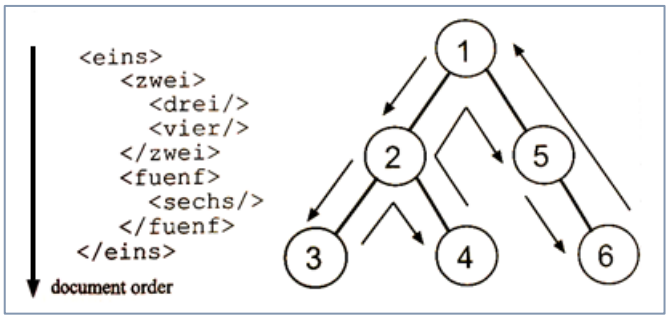

Abbildung 3: Dokumentreihenfolge bei XPath-Ausdrücken (Bongers, F., XSLT 2.0 & XPath 2.0, 2008, S. 257)

Auch schnellere Zugriffe sind jedoch möglich, wie beispielsweise bei der BaseX-Datenbank, die einen *Path Index* nutzt. Dieser wird anfangs aufgebaut, indem das Dokument zu Beginn einmal durchlaufen wird.[43]

Achsen

Bedeutend für die Navigation sind auch der Ausgangspunkt sowie die Richtung, die im Pfadausdruck angegeben werden. Dieser Kontext wird während der Verarbeitung stets neu bewertet und wird deshalb als *dynamischer Kontext* bezeichnet. [44]

---

[40] Vgl. *Vonhoegen, H.*, XML, 2018, S. 191.
[41] Der Dokumentknoten wurde weggelassen.
[42] Vgl. *Bongers, F.*, XSLT 2.0 & XPath 2.0, 2008, S. 256.
[43] Vgl. *BaseX Documentation*, Indexes, 2022.
[44] Vgl. *Bongers, F.*, XSLT 2.0 & XPath 2.0, 2008, S. 261.

Vonhoegen beschreibt in seinem umfassenden Werk „XML“ die Funktion von Achsen wie folgt:

> „Mit Hilfe der Achsenbezeichner kann gesteuert werden, ob von den Eltern aus die Kinder besucht werden, von den Kindern aus die Eltern oder die Geschwister. Mit jedem Schritt innerhalb des Knotengewirrs ändert sich dabei der Kontextknoten für den nächsten Schritt. Jede Achse definiert eine Knotenmenge relativ zum Kontextknoten“[45].

Die Navigation mittels XPath in einem XML-Dokument geht immer von einem bestimmten Ausgangspunkt, dem sog. Kontextknoten, aus. Vom Kontextknoten aus startet demnach die Navigation, während zusätzlich die Richtung benötigt wird. Die Richtung wird durch eine *Achse* angegeben. Wenn in eine tiefere Achse navigiert werden soll, wird von der Kind-Achse gesprochen. Diese stellt gleichzeitig die Default-Achse dar und gründet mit der Eltern-Achse, die wieder in eine höhere Ebene navigiert, die wichtigsten Navigationsrichtungen. Insgesamt unterscheidet die XPath-Empfehlung 13 Achsen[46], von welchen in **Tabelle 1** ein Überblick über die für diese Arbeit relevanten Achsen gegeben wird. In der Übersicht werden auch die möglichen abgekürzten Schreibweisen aufgezeigt.

Achse	Abgekürzte Schreibweise	Beschreibung
self	`.`	Liefert den Kontextknoten.
child	(ist Default-Vorgabe)	Liefert die Kinder des Kontextknotens.
parent	`..`	Liefert die Eltern des Kontextknotens, falls vorhanden.
descendant	`//`	Liefert die Nachkommen des Kontextknotens.
ancestor	(keine Abkürzung vorhanden)	Liefert die Vorfahren des Kontextknotens bis hin zum Wurzelknoten.

Tabelle 1: Übersicht ausgewählter Achsenbezeichner (Vgl. Vonhoegen, H., XML, 2018, S. 195 f.)

Knotentests

Die durch die definierte Achse gewählte Knotenmenge kann anschließend durch den Knotentest weiter gefiltert werden. Dabei wird als Kriterium entweder ein bestimmter Knotenname oder Knotentyp angegeben. Als Knotenname wird der entsprechende Elementname angegeben. Sollen Knoten aufgrund ihres Knotentyps identifiziert werden, wird statt dem Elementname eine entsprechende Funktion als Knotentest genutzt, wie z. B. die Knotentest-Funktion `text()`, die alle Textknoten liefert.[47]

---

[45] *Vonhoegen, H.*, XML, 2018, S. 195.
[46] Vgl. *Hedler, M./Montero Pineda, M./Kutscherauer, N.*, Schematron, 2011, S. 34.
[47] Vgl. *Vonhoegen, H.*, XML, 2018, S. 199.

Lokalisierungspfade

Der wichtigste Ausdruckstyp in XPath ist der Lokalisierungspfad, der angibt, welcher Teil der Knotenmenge eines Dokuments ausgewählt und bearbeitet werden soll. Dafür kann eine ausführliche und eine kürzere Schreibweise verwendet werden.

Außerdem können absolute und relative Pfadangaben verwendet werden, um Objekte zu adressieren. Für Pfadangaben werden Schrägstriche (engl. *slash*) verwendet, um die einzelnen Schritte zu trennen, wobei der erste Slash für den Dokumentknoten steht. Während absolute Pfade den Vorteil haben, dass sie unabhängig vom gerade erreichten Kontextknoten verwendet werden können, haben relative Pfade den Vorteil, dass sie die Mehrfachverwendung von XPath-Ausdrücken erleichtern. Die Stufen des Lokalisierungspfads werden dabei jeweils von links nach rechts ausgewertet. In **Tabelle 2** werden beispielhaft die verschiedenen Möglichkeiten von Pfadangaben in abgekürzter Schreibweise erläutert.

Nr.	Pfadangabe	Erläuterung
1	`/weindepot/anbaugebiet/jahrgang`	XPath-Ausdruck mit einem **absoluten** Pfad.
2	`anbaugebiet/jahrgang`	Möglicher XPath-Ausdruck mit **relativer** Pfadangabe. Stellt den entsprechenden Pfadausdruck zu **Nr. 1** dar, wenn `<weindepot>` der aktuelle Kontextknoten ist.
3	`//jahrgang`	XPath-Ausdruck mit absoluter Pfadangabe, die die Knotenmenge aller Elemente mit dem Namen `<jahrgang>` liefert, unabhängig davon, auf welcher Stufe sie sich befinden. Die Abkürzung mit dem doppelten Slash kann auch innerhalb eines Ausdrucks verwendet werden.
4	`//anbaugebiet/*`	Absoluter XPath-Ausdruck mit einer Wildcard (`*`), die alle Kinder des Elements `<anbaugebiet>` liefert.

Tabelle 2: Verschiedene Möglichkeiten der absoluten und relativen Pfadangabe (Vgl. Vonhoegen, H., XML, 2018, S. 192–194)

Der Kontextknoten selbst kann in einem XPath-Ausdruck mit einem Punkt (.) angesprochen werden. Auch Wildcards (*) können anstelle konkreter Namen verwendet werden. Der Wert eines Attributknotens kann aus der Kombination eines @-Zeichens mit dem Attributnamen erfahren werden.[48]

[48] Vgl. *Vonhoegen, H.*, XML, 2018, S. 192–194.

In der ausführlichen Schreibweise werden doppelte Doppelpunkte als Trennzeichen zwischen Achsenbezeichner und Knotentest verwendet. Der Achsenbezeichner `child`, der gleichzeitig der Default-Achsenbezeichner ist, gibt die Richtung an, in die der Knotendurchlauf erfolgen soll. **Tabelle 3** zeigt beispielhaft die gleiche Pfadangabe in ausführlicher und in kurzer Schreibweise.

Hier wird vom Wurzelknoten zu seinem Kind, dem Wurzelelement `<weindepot>` gesprungen, was die Knotenmenge von `<weindepot>` liefert. Im zweiten Schritt geht es weiter zu dessen Kindelement `<anbaugebiet>`, das wieder seine Knotenmenge liefert, worauf anschließend zu dessen Kindelement `<jahrgang>` gesprungen wird, das schließlich die Knotenmenge dieser Elemente liefert.

Schreibweise	**Pfadangabe in XPath**
Ausführlich	`/child::weindepot/child::anbaugebiet/child::jahrgang`
Kurz	`/weindepot/anbaugebiet/jahrgang`

Tabelle 3: Ausführliche und kurze Schreibweise im Vergleich (Vgl. Vonhoegen, H., XML, 2018, S. 193–195)

In **Abbildung 4** wird bei der ausführlichen Schreibweise des XPath-Ausdruckes deutlich, wie ein Lokalisierungspfad aus einzelnen Lokalisierungsstufen zusammengesetzt wird, die jeweils von links nach rechts ausgewertet werden. Dabei besteht jede Lokalisierungsstufe aus drei Teilen: Achsenbezeichner, Knotentest und optional einem Prädikat oder mehreren Prädikaten. Die allgemeine Form lautet `achsenbezeichner::node-test[predicate]*`.[49]

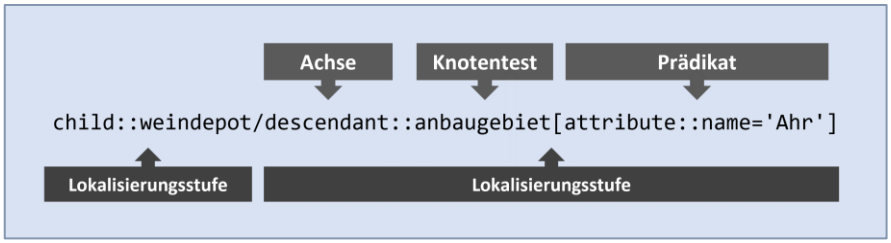

Abbildung 4: Lokalisierungspfad in XPath (in Anlehnung an Vonhoegen, H., XML, 2018, S. 194)

Filtern mit Prädikaten

Mit der Formulierung von Prädikaten lässt sich die Filterung der gewünschten Knotenmenge verfeinern. Dabei wird in eckigen Klammern eine Bedingung formuliert, die erfüllt sein muss, damit bestimmte Knoten ausgewählt werden. Werden mehrere Prädikate verwendet, bildet die Knotenmenge, die das Ergebnis des ersten Prädikats ist, die Grundlage für die Prüfung durch das nächste Prädikat und so weiter. Prädikatsausdrücke unterstützen die allgemeinen Vergleichsoperatoren `<`, `>`, `<=`, `>=`, `=` und `!=`. Da die Zeichen `<` und `>` in XML als Begrenzungszeichen für Elemente gelten, müssen diese als entsprechende Entitätsreferenzen umgeschrieben werden, wie `&lt;`

[49] Vgl. *Bongers, F.*, XSLT 2.0 & XPath 2.0, 2008, S. 1112.

(engl. *lower than)* für `<` und `>` (engl. *greater than)* für `>`.[50] Wenn es sich bei den verglichenen Operanden nicht um atomare Werte handelt, so werden sie zunächst atomisiert. Die so entstandenen Sequenzen werden anschließend verglichen, wobei geprüft wird, ob ein beliebiges Itempaar den Vergleich erfüllt. Sobald die Operanden also als Folgen atomarer Werte vorliegen, werden deren Einzelwerte entsprechend des Operators verglichen.

Weiterhin gibt es seit XPath 2.0 Operatoren, die ausschließlich dem Vergleich von atomaren Einzelwerten dienen: `lt`, `gt`, `le`, `ge`, `eq` und `ne`. Frank Bongers merkt in seinem Werk an, dass,

> „[o]bwohl beispielsweise hinter den Operatorsymbolen ‚eq' und ‚=' die gleiche vordefinierte Operatorfunktion arbeitet [...], das Ergebnis eines Einzelwertvergleichs mit ‚eq' unter Umständen ein anderes als bei einem allgemeinen Vergleich mit ‚=' [ist]"[51].

Für die Ausführung eines Einzelwertvergleichs werden die beiden Operanden zunächst atomisiert; falls diese fehlschlägt, wird eine Fehlermeldung erzeugt. Anschließend wird der Vergleich durchgeführt und das Ergebnis als boolescher Wert zurückgegeben.

Der Unterschied zwischen einer allgemeinen Vergleichsoperation und dem Vergleich von Einzelwerten besteht darin, dass bei zweiterem keine implizite Typumwandlung durchgeführt wird. Außerdem werden bei einem allgemeinen Wertvergleich die Bedingungen bereits „als erfüllt betrachtet, wenn der entsprechende Einzelwertvergleich der atomisierten Werte erfüllt ist".[52]

Mathematische Operatoren wie z. B. `+`, `-`, `*` oder `div` können auch im Prädikat eingesetzt werden, was die Folge hat, dass das Resultat des String-Ausdrucks von einem String-Wert in einen Zahlenwert umgewandelt wird. Auch Klammern dürfen verwendet werden.[53]

Die Bedingung des Prädikats kann in Form eines Vergleichs, eines existenziellen Ausdrucks oder als numerisches Prädikat vorliegen, wie **Tabelle 4** zeigt.

---

[50] Auch die Zeichen `"`, `'` und `&` sind in XML reserviert für die Begrenzung von Attributwerten bzw. Entitäten. Für diese Zeichen sind ebenso entsprechende Entitätsreferenzen als Ersatzdarstellung zu verwenden (vgl. *Becher, M.*, XML, 2021, S. 15 f.).
[51] *Bongers, F.*, XSLT 2.0 & XPath 2.0, 2008, S. 269.
[52] *Ebd.*, S. 271.
[53] Vgl. *Vonhoegen, H.*, XML, 2018, S. 200.

Art des Prädikats	XPath-Ausdruck-Beispiel mit Prädikat	Bedingung ausformuliert
Vergleich	`buch[autor='James Joyce']`	Ein Element `<buch>`, das Kind des Kontextknotens ist, muss ein Kindelement autor besitzen, dessen String-Wert `James Joyce` ist.
Existenzieller Ausdruck	`buch[autor]`	Ein Element `<buch>`, das Kind des Kontextknotens ist, muss ein Kindelement `<autor>` besitzen.
Numerisch	`buch[5]`	Das fünfte Item der Sequenz wird ausgewählt, während die übrigen verworfen werden.

Tabelle 4: Arten von Prädikaten (vgl. Bongers, F., XSLT 2.0 & XPath 2.0, 2008, S. 322 f.)

Bei diesen Beispielen wird also jeweils ein bzw. mehrere `<buch>`–Elementkonten adressiert und nicht etwa `<autor>`. Dies ist auch bei Schematron-Prüfungen relevant, da die adressierten Knoten im Fehlerreport ausgegeben werden bzw. im Oxygen XML Editor gekennzeichnet werden.

2.2.1.2 XPath-Funktionen

Für die Verarbeitung von XPath-Ausdrücken sind innerhalb eines XSLT-Stylesheets sowie auch eines Schematron-Schemas verschiedene Funktionen einsetzbar, die für Zählungen, Berechnungen und zur Untersuchung und Manipulation von Knoten, Zeichenketten und vielem mehr verwendet werden können. Dabei werden einer Funktion beim Aufruf Argumente übergeben, deren Anzahl, Reihenfolge und Werttypen für jede Funktion genau definiert sind. Eine typische XPath-Funktion ist nach dem Muster `funktionsname(argument1, argument2)` aufgebaut. Der Rückgabewert wird an den Ausdruck zurückgeliefert, in dem er aufgerufen wurde. Der Datentyp eines übergegebenen Arguments wird von der Funktion ggf. in den Datentyp konvertiert, der für die Ausführung der Funktion gefordert wird, wie beispielsweise in einen booleschen Wert. Sollte dies nicht möglich sein, so wird ein Fehler gemeldet.[54]

Eine Auswahl der verwendeten Funktionen im Schematron-Schema mit Erklärungen aus der Online-Referenzliste über XPath-Funktionen von W3Schools ist in **Tabelle 5** abgebildet.[55] In der Tabelle werden die Funktionen mit dem Namensraum-Präfix `fn` angegeben, was jedoch in Schematron nur optional anzugeben ist. Das liegt daran, dass

[54] Vgl. *Bongers, F.*, XSLT 2.0 & XPath 2.0, 2008, S. 322–338.
[55] https://www.data2type.de/xml-xslt-xslfo/xslt/xslt-und-xpath-referenz/alphabetische-liste

dieser Namespace unter `http://www.w3.org/2005/xpath-functions` der Default-Namensraum für Standardfunktionen ist und daher üblicherweise weggelassen wird.[56]

XPath-Funktion	Zweck und Rückgabewert
`fn:contains(string1,string2)`	Test auf Substring im Prüfstring: Gibt *true* zurück, falls *string1* den *string2* enthält. Andernfalls gibt der Aufruf *false* zurück.
`fn:count(item,item,...)`	Knotenzahl in der Nodesequenz: Gibt die Anzahl der Knoten als Zahl zurück.
`fn:exists(item,item,...)`	Test auf Vorhandensein der Sequenz: Gibt stets *true* zurück, wenn der Wert der Sequenz **nicht** eine leere Sequenz ist. Andernfalls gibt der Aufruf *false* zurück.
`fn:matches(string,pattern)`	Test auf Patternmatching: Gibt *true* zurück, falls auf das String-Argument das Muster (ein regulärer Ausdruck) passt. Andernfalls gibt der Aufruf *false* zurück.
`fn:not(arg)`	Boolesche Negation: Drückt die boolesche Negation eines übergebenen Wertes aus und gibt *true* zurück, falls der Boolesche Wert *false* ist. Andernfalls gibt der Ausruf *false* zurück, falls der Boolesche Wert *true* ist.

Tabelle 5: Übersicht ausgewählter XPath-Funktionen
(vgl. W3Schools, XPath, XQuery, and XSLT Function Reference, 2023)

2.2.2 XSLT (Extensible Stylesheet Language Transformations)

XSLT ist eine Sprache zur Transformation von XML-Dokumenten, die darauf abzielt, ein Ausgabedokument zu erzeugen. Es ist eine deklarative Programmiersprache, die Programmierkonstrukte wie Sequenzen, Bedingungen und Schleifen in ihrem Vokabular enthält. Zur Ausführung von XSLT transformiert ein XSLT-Prozessor eine XML-Quellinstanz in eine XML-Zielinstanz. Mit sog. Templates werden Regeln definiert, die diese Transformation steuern. Um in den Templates Elemente, Attribute und Textinhalte zu adressieren sowie Funktionen auszuführen, wird XPath verwendet.[57]

[56] Vgl. *Siegel, E.*, Schematron, 2022, S. 177.
[57] Vgl. *Hedler, M./Montero Pineda, M./Kutscherauer, N.*, Schematron, 2011, S. 14 f.

Im Rumpf einer Template-Regel werden Ausgaben erzeugt und es wird die weitere Abarbeitung des Dokumentbaums gesteuert. Die Ausgaben werden beispielsweise durch folgende Komponenten generiert:

- einfacher Text
- das Notieren von Elementen der Zielsprache
- das Erzeugen von Textknoten bzw. Attributwerten aus dem Quelldokument.

Alle XSLT-Elemente und -Attribute müssen dem Namensraum entsprechen, damit der XSLT-Prozessor sie als solche erkennt und verarbeiten kann. Das Namensraum-Präfix `xsl` wird üblicherweise vor die Elemente notiert und gibt damit an, dass sich die Elemente auf den Namenraum unter `http://www.w3.org/1999/XSL/Transform` beziehen. Insgesamt existieren in XSLT 2.0 49 Elemente.[58]

In einem Schematon-Schema eignet sich die Verwendung von XSLT-Anweisungen zur Ausgabe von Textknoten und Attributknoten des Quelldokuments. In den XSLT-Anweisungen werden XPath-Ausdrücke verwendet, um z. B. den adressierten Wert als String in das Ausgabedokument einzufügen. In **Tabelle 6** ist eine Übersicht mit Erklärung der für diese Abschlussarbeit besonders relevanten XSLT-Elemente dargestellt.

Verwendete XSLT-Elemente	**Erklärung**
`<xsl:analyze-string>`	Untersucht einen Eingabe-String, der im Attribut `select` ausgewählt wurde, anhand eines regulären Ausdruckes, der sich im `regex`-Attribut befindet.[59]
`<xsl:if>`	Dient zur Formulierung von Bedingungen, die entscheiden, ob der in der Instruktion enthaltene Templateblock ausgewertet oder übersprungen wird. Es besitzt ein obligatirisches `test`-Attribut.[60]
`<xsl:matching-substring>`	Ist immer ein Kindelement von <`xsl:analyze-string`>.[61]
`<xsl:value-of>`	Wandelt eine im `select`-Attribut angegebene Sequenz in Werte um, die als Output in das Ergebnisdokument kopiert werden.[62]
`<xsl:variable>`	Definiert eine Variable, die mit einem Namen als Referenz versehen wird und der Werte, Strings oder Code zugeordnet werden kann.[63]

Tabelle 6: Übersicht ausgewählter XSLT-Elemente (vgl. Bongers, F., XSLT 2.0 & XPath 2.0, 2008)

[58] Vgl. *Becher, M.*, XML, 2021, S. 290 f.
[59] Vgl. *Bongers, F.*, XSLT 2.0 & XPath 2.0, 2008, S. 672–674.
[60] Vgl. *ebd.*, S. 769.
[61] Vgl. *ebd.*, S. 794.
[62] Vgl. *ebd.*, S. 964.
[63] Vgl. *ebd.*, S. 970.

2.3 Reguläre Ausdrücke

Reguläre Ausdrücke sind standardmäßig in vielen Programmiersprachen wie etwa Perl, PHP oder Java sowie in vielen Werkzeugen eingebaut, wobei es in den Sprachen und Werkzeugen Unterschiede bei der Behandlung regulärer Ausdrücke gibt, die sich z. B. in Dialekten zeigen.[64] Ein regulärer Ausdruck, auch als *Regex* (engl. *regular expression(s)*) bezeichnet, ist nach Sal Manganos Beschreibung „ein String, der ein Muster kodiert, das in einem anderen String gefiltert werden kann".[65] Friedl beschreibt reguläre Ausdrücke als ein „effizientes Mittel, um Texte zu bearbeiten. Reguläre Ausdrücke im engeren Sinne sind eine generelle Notation zur Beschreibung von Textmustern, beinahe wie eine kleine Programmiersprache zum Prüfen und zum Manipulieren von Texten"[66].

In XSLT 2.0 bilden sie ein leistungsstarkes Werkzeug und werden vor allem angewandt, um Textmuster zu erfassen und zu filtern. Hier bietet sich besonders die Nutzung der XPath-Funktion `fn:matches` an, um die Filtermöglichkeiten von XSLT auf den Text eines Knotens zu erweitern. Die XSLT-Instruktion `<xsl:analyze-string>`, erlaubt es, aufwendige Textverarbeitungsanweisungen mittels regulärer Ausdrücke auszuführen.[67]

Größtenteils sind reguläre Ausdrücke in XPath an die entsprechende Syntax aus der Programmiersprache Perl angelehnt, jedoch gibt es dabei diverse Vereinfachungen und damit Abweichungen.[68] In **Tabelle 7** folgt eine Übersicht ausgewählter Regex-Metazeichen mit ihrer Bedeutung, die auch im Schematron-Schema dieser Arbeit verwendet werden.

---

[64] Vgl. *Friedl, J. E. F.*, Reguläre Ausdrücke, 2009, S. 151.
[65] *Mangano, S.*, XSLT-Kochbuch, 2006, S. 31.
[66] *Friedl, J. E. F.*, Reguläre Ausdrücke, 2009, S. 31.
[67] Vgl. *Mangano, S.*, XSLT-Kochbuch, 2006, S. 215.
[68] Vgl. *Bongers, F.*, XSLT 2.0 & XPath 2.0, 2008, S. 513.

Meta-zeichen	Art	Bedeutung
\d	Zeichenklasse	\d entspricht einer dezimalen Ziffer (engl. *digit*).
\s		\s entspricht jedem Leerraumzeichen (engl. *space*).
\w	Zeichenklasse	\w entspricht einem Buchstaben (auch nicht-lateinische Buchstaben), einer Ziffer oder einem Unterstrich (engl. *word character*).
\p{Namen}	Zeichenklasse	\p{Namen} entspricht einem Zeichen einer allgemeinen Unicode-Kategorie oder einem Unicode-Block, der nach einem Namen angegeben ist, wie z. B. Lu für *letter, uppercase* für *Großbuchstabe*.
[]	Zeichenklasse	Entspricht jedem einzelnen Zeichen, dass sich in den eckigen Klammern befindet. So können z. B. auch Zeichenlisten und -bereiche angegeben werden.
?	Quantifizierer	Das vorangehende Zeichen kommt kein- oder einmal vor.
+	Quantifizierer	Das vorangehende Zeichen kommt mindestens einmal vor.
*****	Quantifizierer	Das vorangehende Zeichen darf beliebig oft, auch keinmal vorkommen.

Tabelle 7: Übersicht ausgewählter Regex-Metazeichen
(vgl. .NET, Sprachelemente für reguläre Ausdrücke – Kurzübersicht, 2022)

Ein Backslash gibt in einem regulären Ausdruck an, dass es sich mit dem darauffolgenden Zeichen um ein *Escapezeichen* handelt. Dabei soll eben dieses Zeichen als ein Metazeichen mit der entsprechenden, wie in der Tabelle beschriebenen Bedeutung, interpretiert werden. Ohne den Backslash würde das Zeichen einfach als dieses Zeichen interpretiert werden. Der Backslash hat auch weitere Bedeutungen, auf die hier nicht weiter eingegangen wird.[69]

2.4 Schemasprachen

Oft lohnt es sich, die formale Prüfung auf Wohlgeformtheit um eine Prüfung auf Gültigkeit zu ergänzen, was durch den Einsatz von Schemasprachen erfolgen kann. Dabei geht es um die Frage, ob das Dokument alle von der Anwendung geforderten Informationen enthält und ob diese in gewünschter Reihenfolge erscheinen.[70] Grammatikbasierte Schemasprachen wie DTD, XML Schema oder RELAX NG regeln, „wo und in welcher Reihenfolge Elemente auftreten dürfen, wie oft sie vorkommen, von welchem Datentyp sie sind und welche Attribute sie enthalten“[71]. Auch die Schemasprache Schematron spielt in der Praxis der Datenvalidierung eine wichtige Rolle und ist wie die drei zuvor genannten Schemasprachen Teil der *Document Schema*

[69] Vgl. *.NET*, Sprachelemente für reguläre Ausdrücke – Kurzübersicht, 2022.
[70] Vgl. *Vonhoegen, H.*, XML, 2018, S. 25 f.
[71] *Hedler, M./Montero Pineda, M./Kutscherauer, N.*, Schematron, 2011, S. 11.

Definition Languages (DSDL).[72] Mit Schematron lassen sich logische Beziehungen zwischen Informationseinheiten eines XML-Dokuments beschreiben und prüfen.

2.4.1 DTD (Document Type Definition)

Eine *Document Type Definition* legt restriktiv den Wortschatz und die Grammatik für eine Auszeichnungssprache fest. In einem DTD-Dokument sind die erlaubten Elemente und Attribute deklariert, die im entsprechenden Dokument gültig bzw. typgültig sind. Zum einen behandelt die Deklaration die Bezeichner der Elemente bzw. Attribute und zum anderen das Inhaltsmodell, das festlegt, welche Inhalte in welcher Reihenfolge oder Häufigkeit auftreten dürfen. Zwar ist DTD nicht XML-kompatibel, was u. a. dazu führte, dass im Jahr 2001 XML Schema eingeführt wurde, das denselben Aufgabenbereich abdeckt, jedoch wird es aufgrund der weiten Verbreitung und teilweise einfacheren Anwendung weiterhin in Gebrauch bleiben.[73]

Auch bei ONIX definiert eine DTD diese Klasse von Dokumenten, die alle vom gleichen Typ sind. Da jedes ONIX-Dokument somit mit einem bestimmten Datenmodell übereinstimmt, werden ONIX-Dokumente auch als Instanz eines Modells betrachtet.[74]

2.4.2 Schematron

Schematron ist eine Schemasprache, die zur kontextbezogenen Validierung von Struktur und Inhalt von XML-Dokumenten eingesetzt wird und somit eine Ergänzung zu anderen Schemasprachen darstellt. Es können Regeln formuliert und geprüft werden, die beispielsweise durch DTDs und XSDs nicht abgedeckt werden. Die Schemasprache eignet sich auch für kundenspezifische Regeln.[75]

Der Entwickler von Schematron, Rick Jelliffe, der die Sprache als ein privates Softwareprojekt erstmals 1999 vorstellte, definiert Schematron auf seiner Website prägnant als eine Sprache, mit welcher Aussagen über das Vorhandensein oder Nichtvorhandensein von Mustern in XML-Dokumenten getroffen werden können.[76] Die einfache Implementierbarkeit durch eine zweistufige XSLT-Transformation führte noch vor der Standardisierung zu einer breiten Akzeptanz und einer weiten Verbreitung der Sprache. Seit 2006 ist Schematron ein ISO-Standard, nachdem für die Standardisierung auch die optionale Nutzung weiterer Abfragesprachen neben XPath zur Formulierung von Business Rules verbessert wurde. Heute wird sie von den gängigen XML-Editoren

[72] Vgl. *Vonhoegen, H.*, XML, 2018, S. 73.
[73] Vgl. *Bongers, F.*, XSLT 2.0 & XPath 2.0, 2008, S. 1096.
[74] Vgl. *Vonhoegen, H.*, XML, 2018, S. 72.
[75] Vgl. *Pantopix*, DTD & Schematron vs. XML Schema, 2021.
[76] Vgl. *Jelliffe, R.*, Schematron, 2023.

unterstützt.[77] Auch im Oxygen XML Editor ist die ISO-Version von Schematron standardmäßig implementiert.[78]

Da Schematron in seiner Kompaktheit keine besonderen technischen Hürden birgt, kann es in den Bereichen Webtechnologien, Publishing und Datenbanken auf vielfältige Weisen verwendet werden.[79] Die Sprache kann z. B. zur Qualitätssicherung in Metadaten(banken) und in technischen Dokumentationen eingesetzt werden.[80]

2.4.2.1 Funktionsweise

Die Syntax von Schematron verwendet neben XML eine Untermenge von XPath, das reichhaltige Möglichkeiten zur Analyse und Überprüfung verschiedener Kriterien in XML-Dokumenten bietet. Auf Fehler und Unstimmigkeiten wird aufmerksam gemacht, indem mit eigens formulierten Fehlermeldungen reagiert wird. Mit gefundenen Fehlern sind i. d. R. Items gemeint, die die Fehlerreports ausgelöst haben.

Um ein einfaches Schematron-Schema zu schreiben, benötigt es fünf Elemente. Schematron bedient sich zudem am Business-Rule-Prinzip, was heißt, dass Bedingungen an ein XML-Dokument gestellt und anhand dieser Business Rules überprüft werden können.[81] Fehlermeldungen oder Hinweise werden im Hintergrund als Output generiert und in ein *SVRL-Dokument* (Schematron Validation Report Language) übernommen. Dies ist eine medienneutrale XML-Schnittstellensprache, die es ermöglicht, dieses Protokoll mithilfe einer weiteren Transformation in eine beliebige medienspezifische XML-Syntax zu überführen. Außerdem sieht SVRL Strukturen vor, die neben den eigentlichen Fehlermeldungen noch weiterführende Informationen abbilden können, was ein anpassbares, detailliertes Abbilden der verschiedenen Fehler ermöglicht, wie etwa das Darstellen verschiedener Schweregrade.[82]

Grundsätzlich verwendet Schematron Regeln in Form von `<rule>`-Elementen, die auch mehrere Business Rules beinhalten können. Eine Regel beschreibt bestimmte Kriterien für einen bestimmten Kontext, auf welche das Dokument geprüft wird. Dieser Kontext muss immer festgelegt werden, wobei es sich um einen Kontextknoten, aber auch um eine Kontextknotenmenge handeln kann. Der oder die Kontextknoten werden als XPath-Ausdruck angegeben. Die zu überprüfenden Kriterien werden als Tests definiert, die einer Regel zugeordnet werden. Dazu wird ein Text für eine Fehlermeldung formuliert, die ausgegeben wird, falls ein Test fehlschlägt. Wenn die Position des gefundenen Fehlers angegeben wird, bezieht sich das jeweils auf den Kontextknoten, der im `<rule>`-

---

[77] Vgl. *Hedler, M./Montero Pineda, M./Kutscherauer, N.*, Schematron, 2011, S. 47.
[78] Vgl. *Synchro Soft SRL*, Oxygen XML Editor 24.1, 2020, S. 1192.
[79] Vgl. *Hedler, M./Montero Pineda, M./Kutscherauer, N.*, Schematron, 2011, S. 91.
[80] Vgl. *Pantopix*, DTD & Schematron vs. XML Schema, 2021.
[81] Vgl. *Hedler, M./Montero Pineda, M./Kutscherauer, N.*, Schematron, 2011, S. 47–50.
[82] Vgl. *ebd.*, S. 117 f.

Element im XPath-Ausdruck angegeben ist. Wie es bereits am Ende des **Abschnitts 2.2.1.1** angedeutet wurde, wird beispielsweise im Oxygen XML Editor bei einem Fehlerreport das entsprechende Element, in dem ein Fehler gefunden wurde, mit einer Unterschlängelung markiert.[83]

2.4.2.2 Elemente

Jedes Schematron-Schema beinhaltet das Wurzelelement `<schema>` mit der dazugehörigen Namensraumangabe `http://purl.oclc.org/dsdl/schematron`, um die Sprache ISO-Schematron zu identifizieren. In **Tabelle 8** wird aufgezeigt, wie die wichtigsten Elemente in Schematron lauten, welche Attribute obligatorisch sind und welchen Zweck die Elemente erfüllen.[84]

Element	Obligatorisches Attribut	Zweck
`<schema>`		Wurzelelement
`<pattern>`		Ein Pattern enthält eine oder mehrere Regeln als Kinder. Die `<pattern>`-Elemente werden aneinandergereiht, sind gleichberechtigt und dienen der Gliederung mehrerer Regeln.
`<rule>`	`context="..."`	Hier wird ein Kontextknoten oder eine Kontextknotenmenge in Form eines XPath-Ausdrucks als Attributwert angegeben. Das Element ist Elternelement eines `<report>`–Elements oder mehrerer.
`<report>`	`test="..."`	Hier wird als Attributwert der Test definiert und die Fehlermeldung als Elementinhalt verfasst. Ein Report wird ausgelöst, wenn der Kontextknoten aus dem `<rule>`–Element die Bedingung aus dem `<report>`–Test nicht erfüllt. Als Folge wird der Fehlermeldungstext ausgegeben.
`<assert>`	`test="..."`	Hier wird, wie bei dem `<report>`–Element, der Test definiert und die Fehlermeldung als Elementinhalt verfasst. In einem `<assert>`-Test wird eine Annahme formuliert. Somit wird eine Meldung ausgelöst, wenn der Kontextknoten die Bedingung erfüllt. Als Folge wird der Fehlermeldungstext ausgegeben.

Tabelle 8: Elemente in Schematron
(Vgl. Hedler, M./Montero Pineda, M./Kutscherauer, N., Schematron, 2011, S. 48–55)

[83] Vgl. *Synchro Soft SRL*, Oxygen XML Editor 24.1, 2020, S. 1192.
[84] Vgl. *Hedler, M./Montero Pineda, M./Kutscherauer, N.*, Schematron, 2011, S. 48–55.

Das <report>-Element ist prinzipiell in der Form

```
<report test="XPath-Ausdruck">Fehlermeldungstext</report>
```

aufgebaut. Wenn der XPath-Ausdruck im test-Attribut den Wert *true* zurückgibt, ist ein Fehler enthalten und die Implementierung gibt den Fehlermeldungstext aus. Alternativ dazu gibt es in Schematron das <assert>-Element, das genau wie das <report>-Element aufgebaut ist, jedoch als Fehlerindikator den Wert *false* hat. Somit lässt sich in Schematron semantisch zwischen einem Fehler (<report>) und einer Annahme (<assert>)(engl. *assert*: behaupten) unterscheiden. Bei beiden Elementen liegt der Unterschied in der Verarbeitung des booleschen Wertes, woraus resultiert, dass jeder <report>-Test sich mit der XPath-Funktion not() in einen identisch funktionierenden <assert>-Test umwandeln lässt und umgekehrt.[85]

Zu erwähnen ist noch das optionale <let>–Element. Hier wird eine Variable mit einem Namen im name-Attribut und einem Wert im Attribut value deklariert, die einen vom Ort der Definition abhängigen, begrenzten Geltungsbereich hat. Die <let>–Variable darf in <schema>, <pattern> und <rule> vorkommen. Sinnvoll ist die Verwendung in <rule>, da der Kontextknoten hier auch relativ angegeben werden darf. Das name-Attribut definiert mit einem String den Variablennamen, der später durch ein $-Zeichen referenziert wird. Im value-Attribut wird mitttels einer XPath-Anweisung der Variablenwert angeben.[86] Da XPath auch mit regulären Ausdrücken arbeiten kann, können diese hier angegeben werden.[87]

---

[85] Vgl. *Hedler, M./Montero Pineda, M./Kutscherauer, N.*, Schematron, 2011, S. 52 f.
[86] Vgl. *ebd*, S. 79.
[87] Vgl. *Siegel, E.*, Schematron, 2022, S. 185.

3 Qualitätskriterien bei der Distribution in der Buchbranche

Es ist eine Herausforderung des Buchhandels, dass auf dem globalen Markt zu jeder Zeit Millionen einzelner Produkte verfügbar sind, die von Zehntausenden verschiedener Herausgeber angeboten werden. Die Bestell- und Bestandsauffüllung erfolgt i. d. R. häufig und mit kleinen Änderungen. Die Schaffung einer nachhaltigen Lieferkette für den Buchhandel erfordern Aufmerksamkeit, Planung und Zusammenarbeit zwischen allen Parteien. Aus diesem Grund wurden z. B. neben der Erstellung der ISBN auch weitere Standards und Formate für die Bereitstellung von Daten entwickelt, wie z. B. das von den Branchengremien EDItEUR, BISG und BIC entwickelte Austauschformat ONIX.[88]

Im „Gabler Wirtschaftslexikon“ wird Qualitätssicherung wie folgt definiert: „Die Qualitätssicherung umfasst als Bestandteil des Qualitätsmanagements alle organisatorischen und technischen Maßnahmen, die vorbereitend, begleitend und prüfend der Schaffung und Erhaltung einer definierten Qualität eines Produkts oder einer Dienstleistung dienen“[89]. Dazu gehören folgende Aspekte:

- Beschaffung (Kooperation, Einkauf)
- Qualität (Agilität, Service Level Agreement)
- Total Quality Management (Qualitätsmanagement, Kundenzufriedenheit)
- Produktion (Dienstleistungen, Wirtschaft, Produktionsfaktoren)
- Wartung (Smart Maintenance, vorbeugende Instandhaltung).[90]

In dieser Arbeit wird der Fokus v.a. auf die Wartung, also die Pflege der Metadaten, und die Qualität gesetzt. Die Qualität der Daten hat dabei auch wirtschaftliche Folgen, da sie das Kaufverhalten der Endverbraucher beeinflusst und qualitativ schlechtere Metadaten ggf. Arbeitsaufwand bedeuten, solange sie überhaupt entdeckt werden. Es wird auch der Aspekt der Kundenzufriedenheit betrachtet, was für alle Akteure im Distributionsprozess gilt. So ist es z. B. im Sinne der Autoren und der Verlagspartner, dass Biografieangaben auf einer Website korrekt wiedergegeben werden.

Durch den zwar bereits seit Ender der 1990er Jahren bestehenden[91], aber in den letzten Jahren stark expandierten Onlinebuchhandel zeigt sich, dass sich die Erfolgsfaktoren in der Buchdistribution geändert haben. Dies gilt für besonders für den elektronischen Handel sowohl mit physischen Büchern als auch mit elektronischen Publikationen in Form von E-Books durch das Aufkommen digitaler Marktplätze. Während im stationären Einzelhandel eine Knappheit des Angebots als Vorfilter für den Endkunden besteht und

[88] Vgl. *Walter, D.*, Nielsen Book US Study: The Importance of Metadata for Discoverability and Sales, 2016.
[89] *Voigt, K.-I.*, Definition: Qualitätssicherung, 2018.
[90] Vgl. *Voigt, K.-I.*, Definition: Qualitätssicherung, 2018.
[91] Vgl. *Hiller, S.*, Buchhandelsstrategien im digitalen Markt, 2016, S. 91.

persönliche Beratungen eine Orientierungshilfe boten, wird im Onlinebuchhandel die Sortimentsauswahl und Knappheit nicht mehr direkt ersichtlich und die Beratung wird zum Teil durch Empfehlungsmechanismen ersetzt. Die Präsentation eines Produkts erfolgt im E-Commerce über eine Website des jeweiligen Anbieters und nicht mehr analog auf einer Regalfläche. Außerdem hat der Konsument die Möglichkeit, sich über Kundenrezensionen sowie redaktionell erstellte Beiträge über einzelne Titel zu informieren. Demnach werden die „vormals vom stationären Buchhandel geleisteten Informations-, Beratungs-, Sortiments- und Zahlungsabwicklungsfunktionen […] vom Onlinebuchhandel weitgehend gleichermaßen gewährleistet“[92]. Der Onlinebuchhandel übt durch die Zusammenlegung der Informationen und Inhalte auf einer Plattform eine Handelsfunktion im Sinne der Auffindbarkeit und Präsentation aus, die die für den Kaufprozess entscheidenden Informationen bereitstellt.[93]

Auch auf der Prozessebene stellt der effiziente Onlinebuchhandel andere Anforderungen an die Qualität als der stationäre Buchhandel. Während hier die Wertschöpfung z. B. durch die Standortauswahl, Einkaufsatmosphäre und Veranstaltungen wie etwa Lesungen geprägt wird, sind die Kernaktivitäten in der Online-Distribution andere: Wichtig ist die Implementierung einer nutzer- und kundenfreundlichen Webseite, die eine elaborierte Suchfunktion verfügt. Neben der Effizienz der Logistik, insbesondere wegen der Geschwindigkeit der Lieferung, ist das Marketing in diesem Bereich von großer Bedeutung, da der Bekanntheitsgrad eine wichtige Komponente darstellt. Außerdem sorgt die infolge gesunkener Suchkosten gestiegene Transparenz auf Konsumentenseite dafür, dass es den Verbrauchern erleichtert wird, auch Nischenprodukte ausfindig zu machen, die sie möglicherweise sonst nicht entdeckt hätten. Ein Vorteil des Onlinebuchhandels ist, dass Backlisttitel[94] einen großen Teil seines Umsatzes generieren und auch Nischentitel aufgrund steigender Titelzahl zu einem Umsatzwachstum beitragen.[95] Wenn Nischen- oder Backlisttitel ins Zentrum der Verbraucheraufmerksamkeit rücken, kann davon ausgegangen werden, dass korrekt dargestellte Produktinformationen die Kaufentscheidung positiv beeinflussen und ggf. eine positive Bewertung und Weiterempfehlung begünstigen. Umgekehrt bedeutet das auch, dass unzureichend dargestellte Produktinformationen bei der großen Anzahl an Backlisttiteln möglicherweise keinem Metadatenmanager auffallen, weshalb sich eine teilautomatisierte Prüfung der Daten anbietet.

---

[92] *Hiller, S.*, Buchhandelsstrategien im digitalen Markt, 2016, S. 71.
[93] Vgl. *ebd.*, S. 71 f.
[94] Eine Backlist besteht aus lieferbaren Buchprodukten, die nicht neu erschienen sind.
[95] Vgl. *ebd.*, S. 92.

3.1 Relevanz guter Metadaten

Metadaten sind strukturierte Daten, die dazu dienen, Ressourcen jeglicher Art wie z. B. Daten, Dokumente, Produkte und Konzepte einheitlich zu beschreiben. Sie sorgen dafür, dass den Beschreibungen von Instanzen einer gemeinsamen Ressourcenklasse eine einheitliche Struktur zugrunde liegt und dass dadurch das Suchen, Finden und Selektieren relevanter Ressourcen aus einer Vielzahl von Ressourcen deutlich erleichtert wird. Um Metadaten anwendungsübergreifend nutzen zu können, müssen sie interoperabel sein, was bedeutet, dass sie bestenfalls denselben Metadatenstandard, dasselbe Metadatenmodell und dieselbe Syntax verwenden.[96] Für die Buchbranche ist besonders der Metadatenstandard ONIX for Books mit seinem XML-basierten Dateiformat bedeutend.

Gute Metadaten begünstigen den Verkauf von mehr Büchern. Die VLB[97]-Datenbank des MVB (Marketing- und Verlagsservice des Buchhandels GmbH) besitzt im deutschsprachigen Raum das umfangreichste Verzeichnis zu Buchdaten; im Jahr 2021 waren etwa 2,5 Millionen Titel aus mehr als 22 000 Verlagen gelistet.[98] Beim MVB wird dabei von einer Metadatenbank gesprochen und betont, dass Metadaten richtig und gut gepflegt werden müssen. Regelmäßig beanstandet werden nämlich die Qualität der von den Verlagen gemeldeten Daten. Gute Metadaten erhöhen die Verkaufschancen, indem dem Handel reichhaltige Informationen der Verlage zur Verfügung gestellt werden, darunter auch Verkaufsargumente und Angaben zu Werbemitteln sowie die Angabe von passenden Schlagworten, um bei der Suchmaschinensuche gefunden zu werden.[99] Weiterhin sind Metadaten teilweise da, um sichtbar gemacht zu werden. Aus einem Interviewgespräch mit Barbara Herlinger, einer ONIX-Beauftragten vom Barsortiment „Zeitfracht Medien“ wurde deutlich, dass inkorrekt verwendete Metadatenelemente zusätzlichen Arbeits- und Zeitaufwand nach sich ziehen können, etwa durch Nachfragen bei den Datenzulieferern und die Korrektur der Daten.[100] Dass Metadaten wichtig für die Auffindbarkeit und den Handel sind, zeigt auch das Whitepaper „Nielsen Book US Study: The Importance of Metadata for Discoverability and Sales“ [101] von der US-amerikanischen Nielsen Company. Im Jahr 2016 führte dieser internationale Dienstleister für Verbraucherforschung und andere Bereiche in der Buchbranche eine Studie im US-Markt durch, um die Bedeutung von Metadaten für die Auffindbarkeit und den Vertrieb zu ergründen.

---

[96] Vgl. *Rühle, S.*, Kleines Handbuch - Metadaten, 2012, S. 2–5.
[97] VLB steht für *Verzeichnis Lieferbarer Bücher* und ist eine zentrale Plattform für den automatisierten Austausch von Produktinformationen in der deutschsprachigen Buchbranche.
[98] Vgl. *MVB GmbH*, Über das VLB, 2022.
[99] Vgl. *Heimann, H.*, Metadaten verkaufen mehr Bücher, 2015.
[100] Vgl. *Herlinger, B.*, Zweckentfremdungen in ONIX 2.1 und 3.0, 2022.
[101] *Walter, D.*, Nielsen Book US Study: The Importance of Metadata for Discoverability and Sales, 2016.

Zwar ging es in der Studie um die bereitgestellte Menge an Metadaten und nicht deren Qualität, jedoch kann davon ausgegangen werden, dass die Qualität auch die Distribution beeinflusst. Befinden sich beispielsweise in einem beschreibenden Text wie der Produktbeschreibung Kodierungsfehler, so ist das vergleichbar mit nicht korrekten oder unvollständigen Metadaten, die die Kaufentscheidung beim Endverbraucher negativ beeinflussen können. Gleichzeitig können fehlerhafte Metadaten zu Umständen in Arbeitsprozessen bei den verschiedenen Akteuren entlang der Handelskette führen. Erhalten Dienstleister beispielsweise Metadaten zu Spielen, die sich im Buchhandel oftmals auch im Sortiment befinden, im ONIX-2.1-Format, so müssen die Daten ggf. nachträglich umstrukturiert werden, da ONIX 2.1 nur bedingt Informationsangaben zu Spielen vorsieht.[102]

Korrekte und vollständige, umfassende Buchmetadaten sind außerdem wichtig für die „Discoverability“ (dt. Auffindbarkeit), da sie diese erleichtern. Die Qualität der Auffindbarkeit beeinflusst sowohl Handelspartner als auch Endkunden, die im Internet nach geeigneten Büchern suchen. Gerade bei Belletristiktiteln wurde in der Studie der Nielsen Company beobachtet, dass diese im Durchschnitt öfter verkauft werden, wenn mehr beschreibende Elemente vorhanden sind. Dies kann als Hinweis darauf gewertet werden, dass dieses Genre besonders auf das Surfen und Stöbern nach Produkten, die ihren Bedürfnissen entsprechen, durch Endkunden angewiesen ist. Neben geeigneten Schlüsselwörtern sind reichhaltige beschreibende Daten wichtig für die Entdeckung und Kaufentscheidung. Die Bereitstellung genauer Daten zu Produkteigenschaften wie Seitenzahl und physischer Merkmale hilft bei der Planung der Bestandsverwaltung. Je besser die Vollständigkeit zu den Informationen eines Buchproduktes ist, desto besser kann es sich durch die Lieferkette bewegen und desto öfter wird es nachweislich verkauft. Ausschlaggebende Metadaten zur Vollständigkeit sind nach Einschätzung der Nielsen Company ISBN, Titel, Format bzw. Bindung, Erscheinungsdatum, BISAC-Subjektcode, Endverbraucherpreis, Verkaufsrechte, Titelbild und Mitwirkende. Zusätzlich dazu tragen beschreibende Informationen zur Reichhaltigkeit der Daten bei, wie etwa Titelbeschreibung, die Autorenbiografie und Rezensionen. Bücher, die diese drei beschreibenden Datenelemente beinhalten, erzielten 72 % höhere Verkaufszahlen als Titel ohne diese, wie **Abbildung 5** zeigt.

[102] Vgl. *Herlinger, B.*, Zweckentfremdungen in ONIX 2.1 und 3.0, 2022.

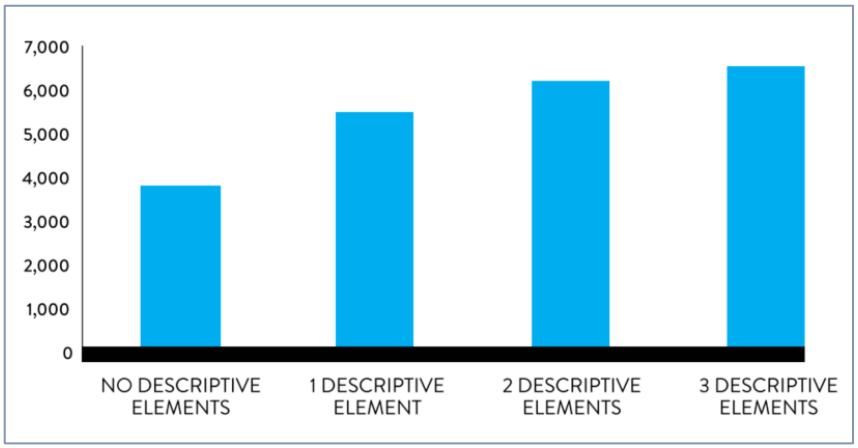

Abbildung 5: Durchschnittlicher Anteil von Käufen von Titeln mit unterschiedlich vielen Beschreibungsdaten (Walter, D., Nielsen Book US Study: The Importance of Metadata for Discoverability and Sales, 2016, S. 9)

Wie bereits erwähnt, kann das Hinzufügen von Keywords die Entdeckungswahrscheinlichkeit eines Titels erhöhen, da sie von Suchmaschinen oder anderen Anwendungen verwendet werden können. Schlüsselwörter können Elemente beinhalten wie z. B. Namen von Charakteren, Orten oder assoziierten Organisationen, behandelte Themen, Begriffe zur groben Beschreibung des Inhalts, verwandte Titel oder ähnliche Autoren.[103]

3.2 Qualitätskriterien von XML-Dokumenten

Bei XML-Dokumenten, die weiterverarbeitet werden müssen, reicht eine bloße Prüfung der Wohlgeformtheit i. d. R. nicht aus. Ein weiteres Qualitätskriterium von XML-Dokumenten, insbesondere bei XML-Anwendungen, ist eine klare Strukturierung der Elemente, um eine systematische Auswertung zu ermöglichen.[104] Bei Prozessschritten rund um Publishing ist die Validierung von Dokumenten ein Freigabekriterium. Dafür werden häufig folgende grammatikbasierte Schemasprachen verwendet:

- DTD, die bei Verlagen häufig anzutreffen ist, jedoch nicht weiterentwickelt wird[105]
- XML Schema (oder kurz *XSD* für *XML Schema Definition*), das ein vom W3C herausgegebener Standard ist
- RELAX NG (*Regular Language for XML Next Generation*), das von OASIS herausgegeben wurde.

Dabei können Abhängigkeiten der Strukturen zu anderen Elementen oder Attributen im Dokument je nach Schemasprache nur teilweise oder gar nicht ausgedrückt werden. Bei der Validierung von XML-Dokumenten sind vor allem zwei Dinge entscheidend: die

[103] Vgl. *Walter, D.*, Nielsen Book US Study: The Importance of Metadata for Discoverability and Sales, 2016.
[104] Vgl. *Grupe, W.*, XML: Schwach strukturierte Dokumente, 2021.
[105] Vgl. *data2type GmbH*, Gültige Dokumente korrekte Dokumente.

Qualität der zugrunde liegenden Regeln und der Umfang bzw. die Möglichkeiten der Sprache, in denen diese Regeln ausgedruckt werden. Während die Qualität der Regeln vom Entwickler dieser abhängt, sind die Möglichkeiten einer (Schema-)Sprache durch den jeweiligen Sprachstandard begrenzt. Der Einsatz von Schemasprachen kann also bereits einen beträchtlichen Teil an Qualität sichern, aber eine einzelne Schemasprache kann nicht 100 % formal beschriebener Regeln abbilden.[106]

Bei XML-Dokumenten können „drei Stufen der Prüfung von XML-Dokumenten"[107] unterschieden werden: die Prüfung der Syntax, der Grammatik und der Kohärenz. Damit XML-Daten alle Anforderungen zur korrekten Weiterverarbeitung möglichst optimal erfüllen, benötigt es Technologien, die alle drei der genannten Prüfungen ermöglichen. Bei der Syntax-Prüfung ist es von Bedeutung, ob die Daten allen Regeln der W3C-Empfehlung vorliegen und das XML-Dokument somit wohlgeformt ist. Um dies sicherzustellen, wird ein XML-Parser eingesetzt. Die Grammatik wird geprüft, indem kontrolliert wird, ob die Daten im XML-Dokument den in einer Schemasprache festgelegten Regeln entsprechen. Diese Regeln sind durch Reihenfolge, Häufigkeit, Hierarchie und Datentyp der XML-Daten definiert, was ein validierender Parser überprüft. Wenn ein XML-Dokument diesen grammatikalischen Regeln entspricht, wird es valide bzw. Schema-valide genannt. Bei der Prüfung der Kohärenz ist relevant, ob die inhaltlichen Beziehungen zwischen den einzelnen Informationseinheiten korrekt sind, was für die Funktionsweise der jeweiligen Anwendung bedeutsam ist. Aus den inhaltlichen Beziehungen lassen sich einschränkende Regeln ableiten, die eingehalten werden sollen. Für die Prüfung der Kohärenz können mit Schematron logische Beziehungen zwischen Informationseinheiten eines XML-Dokuments beschrieben werden und die Strukturen schließlich abhängig vom Kontext eines XML-Fragments ausgewertet werden. Wenn diese Regeln in Schematron formuliert sind und ein Dokument diesen entspricht, kann es Schematron-valide genannt werden.[108] Der Einsatz von Schematron ermöglicht es, zunächst spezifische Business Rules zu formulieren und als Prüfregeln zu verfassen, um ein XML-Dokument daraufhin dagegen zu validieren.

---

[106] Vgl. *Hedler, M./Montero Pineda, M./Kutscherauer, N.*, Schematron, 2011, S. 11.
[107] *Ebd.*, S. 11.
[108] Vgl. *ebd.*, S. 11.

4 ONIX for Books

Das XML-Format ONIX for Books ist eine internationale Standardspezifikation für den Austausch von Produktinformationen innerhalb der Buchbranche. Kurz auch ONIX genannt (von *Online Information Exchange*), ist es ein international etabliertes Austauschformat für die Kommunikation von Buch-, E-Book- und digitalen Audio-Metadaten zwischen Verlagen, Distributoren, Dienstleistern, Händlern sowie weiteren Zwischenhändlern in der Buchlieferkette.

Insgesamt bringt ONIX mehrere Geschäftsvorteile: Es ermöglicht die Bereitstellung umfassender Produktinformationen wie z. B. Buchtitel, Informationen über Autoren und andere Mitwirkende eines Titels, Preis und physische Eigenschaften des Produktes und den Austausch dieser Informationen in einer Standardform innerhalb der Lieferkette, wobei ein einziger Datensatz für alle nachgelagerten Prozesse geeignet ist. Durch die Bereitstellung einer Vorlage für den Inhalt und die Struktur der Informationen über Buchprodukte hat ONIX die Entwicklung besserer interner Informationssysteme angeregt, in welchen alle Metadaten zusammengeführt werden. Diese Metadaten sind bedeutend für die Beschreibung und das Bewerben von Neuerscheinungen und Backlist-Titeln und werden u. a. zur Erstellung von Texten zu Katalogsinformationen, Vorankündigungen oder anderem Werbematerial genutzt.[109]

Da das Dateiformat für die Computer-to-computer-Kommunikation entwickelt und optimiert wurde, ist es für den Menschen eher schwierig zu lesen und zu interpretieren. Unterstützung bieten im Internet zugängliche Dokumentationen zu allen ONIX-Elementen und den sog. Codelisten, die im **Abschnitt 4.2.2** erläutert werden. Weil es um die Kommunikation von Informationen geht, werden die ONIX-Dateien auch als ONIX-Meldungen bzw. *ONIX messages* bezeichnet. Diese ONIX-Meldungen fließen zwischen den (Meta-)Datenbanken, während das ONIX Framework eine standardisierte Terminologie enthält, um sicherzustellen, dass Branchenbegriffe von den ONIX-Nutzern möglichst einheitlich interpretiert und verwendet werden.

Obwohl das Format speziell auf Buchprodukte ausgerichtet ist, wurden auch Metadaten über andere Produkte ausgetauscht, die in der Buchbranche verbreitet sind, wie etwa Lernsoftware, kartografische Produkte, Spiele und elektronische Geräte wie E-Reader. Während die heute für ONIX verantwortliche Organisation EDItEUR auf ihrer Website schreibt:

> „Although the format is formally known as ONIX for Books, it has always covered other media such as audiobooks or recorded video and other products produced by publishers and other organisations which are distributed through the book supply chain“[110],

[109] Vgl. *EDItEUR*, ONIX FAQs.
[110] *Ebd*.

gibt es in der weit verbreiteten ONIX Version 2.1 keine entsprechende Struktur bzw. geeignete Elemente, um reichhaltige Informationen zu Spielen bzw. Nonbookprodukten festzuhalten.

4.1 Entwicklung und Verbreitung von ONIX

ONIX wurde ursprünglich in Zusammenarbeit von der Digital Issues Working Group der Association of American Publishers, EDItEUR und anderen im Jahr 1999 als Reaktion auf die wachsende Bedeutung hochwertiger Metadaten für Verlage und Onlinebuchhändler entwickelt und wurde auch von der XML-Spezifikation beeinflusst, die 1998 vom W3C veröffentlicht wurde. Seit der Veröffentlichung von ONIX 1.0 im Januar 2000 gab es zahlreiche Aktualisierungen des Standards. Nach der ersten Veröffentlichung von Release 2.1 im Jahr 2003 erfuhr ONIX 2.1 kleinere Aktualisierungen und ist seit 2004 stabil sowie zur Vorgängerversion vollständig abwärtskompatibel. Die Version 2.1 ist in vielen Märkten, insbesondere in den USA und in Deutschland, eine häufig implementierte Version. Die Version ONIX 3.0 wurde im April 2009 veröffentlicht und erhält immer wieder Updates, die optionale Ergänzungen zur ONIX-3.0-Funktionalität beinhalten. Da ONIX 3.0 sich in der Struktur und Austauschtechnologie von 2.1 unterscheidet und nicht mehr abwärtskompatibel ist, ist es eine größere Aufgabe für die Lieferkettenpartner, auf die neuste Version umzustellen. Gleichzeitig blieb jedoch mehr als die Hälfte vom ONIX-3.0-Aufbau gegenüber 2.1 im Wesentlichen unverändert. [111]

Seit 2014 wird ONIX 2.1 – mit einer Vorankündigung von drei Jahren – nicht mehr weiterentwickelt und der Support durch EDItEUR wurde stark reduziert. EDItEUR ist eine internationale Organisation, die die Entwicklung und Förderung von Standards im digitalen Bereich im Buch- und Zeitschriftensektor koordiniert und eine Vielzahl von Dokumenten zu ONIX, wie auch die Spezifikationen und Codelisten, online zur Verfügung stellt.[112] Obwohl EDItEUR mit dem Sunset Date beabsichtigte, die Migration auf 3.0 bei den Organisationen voranzutreiben, die noch Version 2.1 verwendeten, wird 2.1 immer noch viel genutzt. Gründe für die starke Nutzung sind, dass

- es generell nutzbar ist und von EDItEUR geduldet wird[113],
- es den meisten Anforderungen in der Buchbranche weiterhin genügt,
- die Migration auf 3.0 wegen der neuen Struktur aufwendig ist und aus diesem Grund mit Kosten verbunden ist,
- viele Lieferkettenpartner – auch im Ausland – ONIX 3.0 nicht nutzen und dadurch eine Abhängigkeit herrscht.[114]

---

[111] Vgl. *EDItEUR*, Frequently Asked Questions about ONIX for Books, 2009.
[112] Vgl. *EDItEUR*, ONIX for Books: Product Information Message Product Record Format, 2006, S. 2.
[113] Vgl. *EDItEUR*, ONIX FAQs.
[114] Vgl. *Herlinger, B.*, Zweckentfremdungen in ONIX 2.1 und 3.0, 2022.

Laut EDItEUR sind die Versionen 2.1 und 3.0 beide weit verbreitet, während die Verwendung von 3.0 zunimmt.[115] Es wird aber geschätzt, dass ONIX 2.1 zumindest noch einige Jahre verwendet wird.[116]

Die bedeutendsten Gründe für die Einführung von ONIX 3.0 sind die Abdeckung digitaler Produkte mit Raum für neue Produktformate, um mit dem Stand der Technik gehen zu können, sowie die Eliminierung veralteter Elemente. [117]

4.2 Aufbau einer ONIX-Datei in ONIX 2.1 und 3.0

Da ONIX ein XML-Datenformat ist, sind alle darin enthaltenen Daten in XML-Tags eingebettet. Die in ONIX verwendeten Elemente werden laut Spezifikation ONIX-Tags genannt. ONIX-Tags und der Elementinhalt bilden ein ONIX-Datenelement. Alle ONIX-Tags und eine Beschreibung ihrer erlaubten und vorgesehenen Inhalte sind in der Dokumentation der entsprechenden ONIX-Spezifikation nachzulesen, welche jeweils auf der Website[118] von EDItEUR zu finden ist.

Eine Besonderheit von ONIX ist, dass es zwei Möglichkeiten von Elementnamen mit identischer Bedeutung erlaubt, wobei eine Mischung beider Arten in einer ONIX-Datei nicht erlaubt ist. So kann die ONIX-Datei entweder in Short-Tag- oder in Reference-Tag-Syntax ausgezeichnet sein, was **in Abschnitt 4.2.2** genauer behandelt wird. Im Folgenden wird zur Vereinfachung ausschließlich der Aufbau einer ONIX-Datei mit Short-Tags vorgestellt.

4.2.1 Grundlegender Aufbau einer ONIX-Datei

Generell startet eine ONIX-Datei mit einer Reihe von Deklarationen, von welchen zwar keine verpflichtend ist, bezüglich derer jedoch die Spezifikation eine Deklaration eines Schemas wie einer DTD oder XSD dringend empfiehlt. Eine Assoziation mit einem Schematron-Schema ist in ONIX nicht explizit vorgesehen, aber kann z. B. mit einer `xml-model`-Processing-Instruction[119] vorgenommen werden. Diese Verarbeitungsanweisung wird beispielsweise vom Oxygen XML Editor berücksichtigt. Anschließend folgt das Wurzelelement, das die Metadaten in ONIX-Tags bzw. -Elementen enthält.

Im **Listing 2** wird beispielhaft der Aufbau einer typischen ONIX-2.1-Meldung in Short-Tag-Syntax gezeigt, die mit einer XML-Deklaration beginnt und anschließend die Dokument-Typ-Deklaration angibt, welche die DTD definiert, gegen die das ONIX-

[115] Vgl. *EDItEUR*, ONIX FAQs.
[116] Vgl. *Herlinger, B.*, Zweckentfremdungen in ONIX 2.1 und 3.0, 2022.
[117] *EDItEUR*, Frequently Asked Questions about ONIX for Books, 2009.
[118] https://www.editeur.org/8/ONIX/
[119] Vgl. *W3C*, Associating Schemas with XML documents 1.0 (Third Edition), 2012.

Dokument validiert werden soll.[120] Darauf folgt das Wurzelelement `<ONIXmessage>`, das einmalig ein `<header>`-Element enthält, in welchem Informationen zum Austausch der Datei festgehalten sind. Darauf folgt das Herzstück der ONIX-Datei, nämlich die `<product>`-Elemente, die die Container für die Buchmetadaten in Form von ONIX-Elementen darstellen. Ein `<product>`-Element enthält dabei die Metadaten zu einem Buchprodukt.

```
<?xml version="1.0"?>
<!DOCTYPE ONIXmessage SYSTEM
"http://www.editeur.org/onix/2.1/short/onix-international.dtd">
<ONIXmessage>
  <header>
    <!--Message header data elements -->
  </header>
  <product>
    <!--Product information data elements for product 1 -->
  </product>
    <!-- ... -->
</ONIXmessage>
```

Listing 2: Aufbau einer ONIX-2.1-Meldung
(in Anlehnung anEDItEUR, ONIX for Books: Product Information Message XML Message Specification, 2008, S. 7)

Der Grundaufbau in ONIX 3.0 ist grundlegend der gleiche. Allerdings ist das `release`-Attribut im Wurzelelement verpflichtend, während es bei 2.1 optional war, und die Version 3.0 befindet sich seit Ende 2011 in dem Namensraum, der im **Listing 3** bei **[1]** zu sehen ist.[121] Es sind zudem weitere ONIX-Elemente, Elementverbunde bzw. *Composites* dazugekommen oder verändert worden. Wie bereits erwähnt, wurden in ONIX 3.0 veraltete Elemente eliminiert.

```
<?xml version="1.0"?>
<!DOCTYPE ONIXmessage SYSTEM
"https://www.editeur.org/onix/3.0/short/ONIX_BookProduct_3.0_sh
ort.dtd">
<ONIXmessage xmlns="http://ns.editeur.org/onix/3.0/short"          [1]
release="3.0" >
  <header>
    <!--Message header data elements -->
  </header>
  <product>
    <!--Product information data elements for product 1 -->
  </product>
    <!-- ... -->
</ONIXmessage>
```

Listing 3: Aufbau einer ONIX-3.0-Meldung
(in Anlehnung an EDItEUR/Bell, G., ONIX for Books: Summary of changes for ONIX for Books 3.0 revision 1, 2012, S. 10)

---

[120] Vgl. *EDItEUR*, ONIX for Books: Product Information Message XML Message Specification, 2008, S. 19 f.
[121] Vgl. *EDItEUR/Bell, G.*, ONIX for Books: Summary of changes for ONIX for Books 3.0 revision 1, 2012, S. 10.

Während bei der Verwendung von Short-Tag-Syntax das Wurzelelement `<ONIXmessage>` ist, lautet es bei der Verwendung von Reference-Tags `<ONIXMessage>`. So kann nur anhand des Top-Level-Elements erkannt werden, welche Syntax der ONIX-Elemente vorliegt, womit wiederum festgestellt werden kann, welche Art der Schemadateien zur Validierung herangezogen werden muss (*short* oder *reference*). Die ONIX-Version kann anhand des `release`-Attributs erkannt werden: Ist dieses nicht notiert, so handelt es sich um die Version 2.1, da hier dieses Attribut im Gegensatz zu ONIX 3.0 optional ist.[122] In **Abschnitt 4.3** ist ein Entscheidungsbaum dargestellt, der diese Thematik genauer beleuchtet.

Da in der `DOCTYPE`-Deklaration oder bei der `xsi:schemaLocation` i. d. R. auf einen lokalen Speicherort des Schemas verwiesen wird, empfiehlt EDItEUR, diese Schema-Assoziationen vor dem Versenden der ONIX-Meldung an Partner zu entfernen.[123] Alternativ kann, wenn nur die Dateipfade unterschiedlich sind und die Dateinamen gleich sind mit den seitens EDItEUR bereitgestellten Schemadateien, der *systemSuffix*- bzw. *uriSuffix*-Mechanismus verwendet werden, um die empfängerseitig gespeicherten Schemadateien zu verwenden.[124]

4.2.2 ONIX-Tags und Codelists

In ONIX existieren etwa 450 verschiedene ONIX-Elemente, wobei diese sich in folgenden erlaubten Inhalten unterscheiden: Textdaten (z. B. Namen oder Biografietext), Ziffern, Daten mit einem eingeschränkten Wertebereich (sog. *Codes*) wie etwa `A01`, und strukturelle Tags, die andere untergeordnete Elemente gruppieren.

Alle ONIX-Elemente und eine Beschreibung ihrer vorgesehenen Inhalte sind in der Dokumentation der entsprechenden ONIX-Spezifikation nachzulesen, die online und frei verfügbar ist. Datenelemente mit eingeschränkten Wertebereichen bzw. Codes werden in Codeslists definiert. Für verschiedene ONIX-Elemente gibt es aktuell etwa 160 verschiedene Codelisten, welche EDItEUR auch online zur Verfügung stellt. Für diese Arbeit werden folgende Dokumente als Referenz gewählt: Für ONIX 2.1 wird die Spezifikation der „*Revision 03*“[125] von Januar 2006 und die Codeliste „Codelists Issue 36“[126] verwendet, während für ONIX 3.0 die Spezifikation „Revision 8“[127] von Juni 2021

[122] Vgl. *EDItEUR/Bell, G.*, ONIX for Books: Summary of changes for ONIX for Books 3.0 revision 1, 2012, S. 10.
[123] Vgl. *EDItEUR*, ONIX for Books: Product Information Format Specification, 2021, S. 24.
[124] Vgl. *OASIS*, XML Catalogs, 2005, S. 16–19.
[125] *EDItEUR*, ONIX for Books: Product Information Message Product Record Format, 2006.
[126] *EDItEUR*, ONIX for Books: Codelists Issue 36, 2017.
[127] *EDItEUR*, ONIX for Books: Product Information Format Specification, 2021.

und die Codeliste „Codelists Issue 59“[128] betrachtet wird. EDItEUR stellt auch interaktive Codelisten für ONIX 2.1[129] und 3.0[130] zur Verfügung.

Wie bereits erwähnt, ist eine ONIX-Datei entweder in Short-Tag- oder Reference-Tag-Syntax ausgezeichnet. Jedes Element existiert sowohl als längerer Referenzname in Klartext (z. B. `<PersonNameInverted>`) als auch als kurzer Elementname bestehend aus einem Buchstaben und drei Ziffern (z. B. `<b037>`). Im folgenden Beispiel in **Listing 4** drücken die Elemente links und rechts das gleiche aus, während auf der linken Seite die Reference-Tags und rechts die Short-Tags genutzt werden. Abgebildet ist hier das *Contributor composite* (engl. *composite*: Verbund, Zusammensetzung), welches Informationen zu einem Mitwirkenden (engl. *contributor*: Mitwirkender) beschreibt. Das Element `<SequenceNumber>` bzw. `<b034>` mit dem Inhalt `2` bedeutet, dass in diesem Verbund gerade ein zweiter Mitwirkender beschrieben wird. Das Element `<b035>` mit dem Elementinhalt bzw. Code `A01` beschreibt die Rolle des Contributors als Autor, was nach ONIX-Spezifikation in *List 17* angegeben ist. In `<b037>` steht der Name des Autors in Textform in umgekehrter Reihenfolge. Die Elementreihenfolge ist dabei festgelegt und im Schema definiert. Es ist auch definiert, welche Elemente obligatorisch sind; sollte es für Elemente, die nicht verpflichtend sind, keinen Inhalt geben, so werden diese in der Meldung weggelassen.

Wie beispielsweise `<b037>`, sind einige ONIX-Elemente also für freitextlichen Inhalt vorgesehen. Bestimmte ONIX-Elemente, wie `<b044>` und insbesondere `<d104>`, können dabei relativ lange Texte beinhalten. In Metadatenfeldern, die typischerweise Listenstrukturen oder Absätze aufweisen, empfiehlt EDItEUR, diese Texte in XHTML einzubetten. Alternativ kann auch HTML in CDATA oder *escaptes* HTML verwendet werden, wovon EDItEUR jedoch entschieden abrät. Der Einsatz von XHTML zur Auszeichnung von Listen und Absätzen wird u. a. deshalb empfohlen, da XHTML strengere Regeln als HTML besitzt, und bei einer Validierung gegen die DTD oder XSD auch das XHTML validiert wird.[131] Auf diese Thematik wird in **Abschnitt 5.3** genauer eingegangen.

Reference Tags	Short Tags

`<Contributor>`
`  <SequenceNumber>2</SequenceNumber>`
`  <ContributorRole>A01</ContributorRole>`
`  <PersonNameInverted>Badenov, Boris</PersonNameInverted>`
`</Contributor>`
	`<contributor>`
	`  <b034>2</b034>`
	`  <b035>A01</b035>`
	`  <b037>Badenov, Boris</b037>`
	`</contributor>`

Listing 4: Contributor composite in Reference- und Short-Tag-Syntax (vgl. EDItEUR, ONIX FAQs)

[128] *EDItEUR*, ONIX for Books: Codeslists Issue 59, 2022.
[129] https://ns.editeur.org/onix36/en
[130] https://ns.editeur.org/onix/en
[131] Vgl. *EDItEUR*, ONIX for Books: Product Information Message, 2021, S. 2.

Die Short-Tags haben den Vorteil, dass die ONIX-Datei um etwa ein Viertel bis ein Drittel kleiner und prägnanter sind, während Reference-Tags verständlicher für den Leser und somit einfacher für Debugging sind. Für ONIX 2.1 und 3.0 existieren Tagname-Konverter, die in Form einer XSL-Transformation in ONIX-Dateien Reference-Tags in Short-Tags umwandeln können und umgekehrt. Die jeweiligen Entsprechungen von Short- und Reference-Tag-Namen sind in den jeweiligen DTDs oder XSDs[132] als Default-Attribute hinterlegt, so dass diese Attribute nach dem Parsen der XML-Datei explizit vorhanden sind. Insofern ist das benötigte XSLT sehr einfach gehalten, da die Mapping-Information ausschließlich in den DTDs bzw. XSDs steht. Aufgrund der Prägnanz der Short-Tags und der schnelleren bzw. kürzeren Notation der meist vierstelligen Elementnamen wird in dieser Abschlussarbeit die Umwandlung in Short-Tags bevorzugt. Laut EDItEUR wurde die Nutzung der längeren Reference-Tags zunehmend von den Nutzern bevorzugt[133], allerdings zeigte sich bei den für diese Arbeit bereitgestellten ONIX-Dateien keine eindeutige Tendenz.

4.3 Validierung von ONIX-Dateien anhand Schema-Dateien

Für ONIX 2.1 standen ursprünglich Kopien der jeweils neusten Versionen der DTD und XSD online auf der Website von EDItEUR zur Verfügung, die bei einer Validierung referenziert werden konnten. Im Jahr 2014 wurde diese Option entfernt, sodass Nutzer von ONIX 2.1 ihr Schema lokal speichern müssen. Für ONIX 3.0 stehen Schemadefinitionen als XSD, DTD und RELAX NG zur Verfügung, wobei keine der Optionen online referenziert werden kann. EDItEUR empfiehlt zu Validierungszwecken die neueste Version der Schemaoption auf ihrer Website herunterzuladen und lokal zu speichern, damit Validierungen schneller und zuverlässiger sind. Gleichzeitig wird die Verwendung von XSD- und RELAX-NG-Schemas empfohlen und betont, dass die Verwendung einer DTD in erster Linie für die Verwendung mit XML-Tools wie etwa dem Tagname-Konverter vorgesehen ist. In jeder der drei Schemasprachen existiert jeweils eine Schemadatei für Short- und für Reference-Tags.[134]

In **Abbildung 6** ist ein Entscheidungsbaum dargestellt, der dem Empfänger einer ONIX-Meldung helfen kann, das geeignete Schema für die Validierung heranzuziehen. Anhand dieser Erkennungsmerkmale, die einander teilweise ausschließen, könnten weitere Schematron-Konsistenzprüfungen erstellt werden.

---

[132] Vgl. *EDItEUR/Bell, G.*, Using local DTD and XSD files after ONIX 2.1 sunset, 2014.
[133] Vgl. *EDItEUR*, ONIX FAQs.
[134] Vgl. *EDItEUR/Bell, G.*, Using local DTD and XSD files after ONIX 2.1 sunset, 2014.

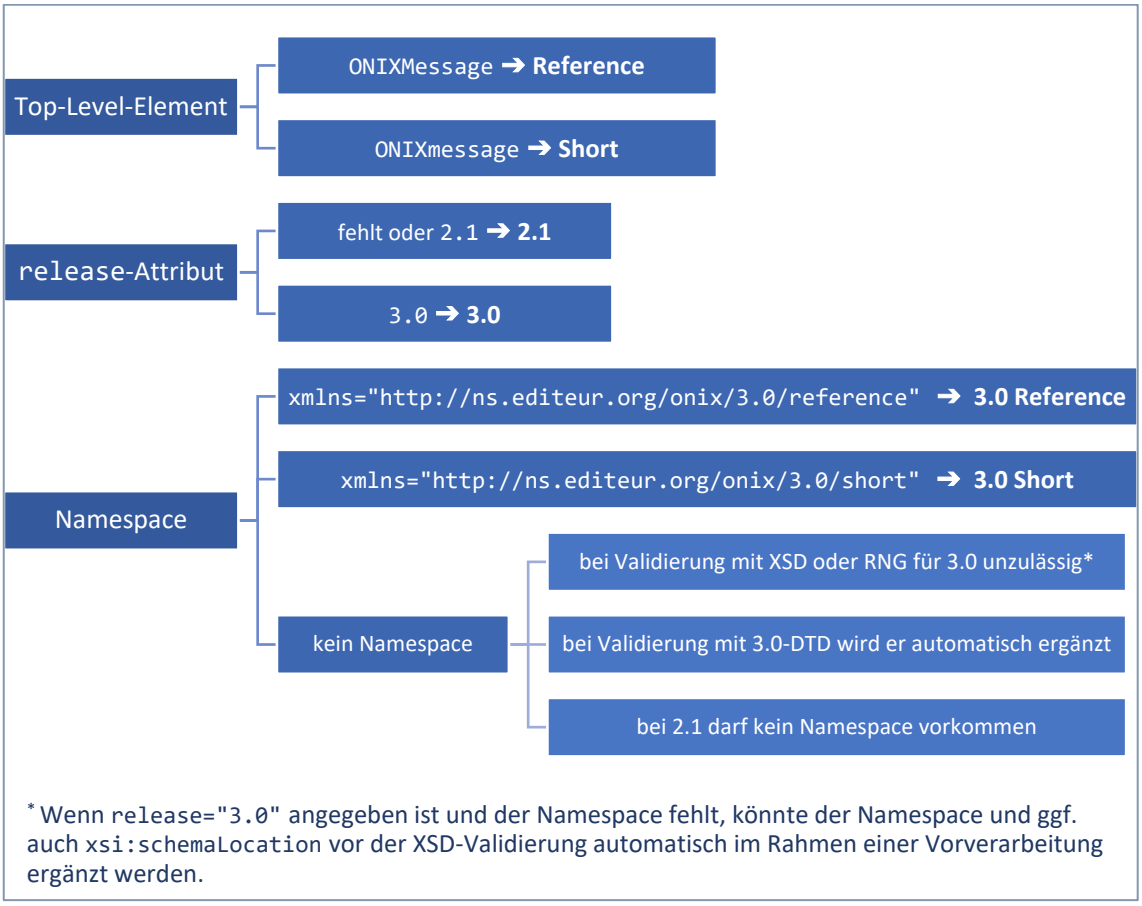

Abbildung 6: Entscheidungsbaum für Schema-Auswahl (eigene Darstellung)

Da Schematron sich gut in XML-Workflows integrieren lässt, bietet es sich an, ein Schematron-Schema für ONIX-Dateien zu entwickeln, das die von EDItEUR bereitgestellten Schema-Dateien ergänzt. Dabei können die Schematron-Prüfregeln an eigene Bedürfnisse angepasst werden.

Für das Beheben von ausgelösten `<report>`- und `<assert>`-Meldungen ist denkbar, die *Schematron QuickFix Language* (SQF) einzusetzen. Mit einer SQF-Erweiterung lassen sich Fehlerbeseitigungen (engl. *fixes*) definieren, die bei einem Fehlerreport am entsprechenden Element anwählbar sind und bei Wunsch den Fehler einmalig automatisch beheben.[135] Allerdings sind Schematron QuickFixes flächendeckend für ONIX-Dateien nur bedingt zu empfehlen. So müssten Metadatenmanager mit der Anwendung dieser Erweiterung vertraut sein und sind in der Wahl der Entwicklungsumgebung beschränkt.[136] I. d. R. werden die ONIX-Dateien aus einem führenden System wie z. B. Klopotek exportiert, ohne eine direkte Möglichkeit zu haben, Korrekturen in dieses System zurückzuschreiben.[137]

---

[135] Vgl. *Kutscherauer, N.*, Schematron QuickFix project, 2018.
[136] Der Oxygen XML Editor enthält eine nutzerfreundliche SQF-Implementierung. Es gibt auch eine Online-Implementierung unter http://escali.schematron-quickfix.com/
[137] Vgl. *Pufe, M.*, Wichtige ONIX-Elemente und Reklamationen, 2022.

Außerdem ist die Umsetzung von Schematron QuickFixes für das Schematron-Schema dieser Arbeit nur für einen kleinen Anteil der Regeln lohnend bzw. gut umsetzbar. Möglicherweise möchten sich Anwender des Schemas auch nur einen Überblick über die Fehlerreports und Problematiken verschaffen. Da es sich um Metadaten handelt, die zwischen verschiedenen Partnern ausgetauscht werden, könnte eine automatische Behebung der Fehler, die individuelle Entscheidungen abnimmt, abschreckend für mögliche Nutzer des Schematron-Schemas sein. Da schwerwiegende Fehler durch die simple Validierung gegen eine ONIX-DTD oder andere Schema-Dateien, die von EDItEUR bereitgestellt werden, zu vermeiden sind, sind fatale Fehler ohnehin ausgeschlossen. Denkbar wäre dennoch, bei Wunsch einzelne Schematron-Prüfregeln um SQF zu erweitern, wenn sie beispielsweise besonders oft vorkommen.

5 Fehler und Zweckentfremdungen in ONIX 2.1

Wie in der Einleitung erwähnt, sollten für diese Arbeit ursprünglich regelmäßig zweckentfremdete ONIX-Elemente aufgespürt werden. Insgesamt fallen Zweckentfremdungen meist durch Reklamationen der Handelspartner auf, die die Daten abnehmen, da der Verarbeitungsprozess weitgehend automatisiert ist und die ONIX-Meldungen nur auf formale Validität geprüft werden[138]. Zweckentfremdete Elemente führen zu einer uneinheitlichen Verwendung von Dateielementen im Austauschformat und können zu Fehlern, zu Mehraufwand in der Verarbeitung der Daten und zu fehlerhaften Angaben beispielsweise auf Websites führen. Resultieren können die Zweckentfremdungen zum einen daraus, dass in ONIX 2.1 die Möglichkeiten für gewisse Angaben nicht bestehen (z. B. ausreichende Angaben für Marketinginformationen), die Metadatenmanager deswegen nach Kompromissen suchen und die Daten in ein „verwandtes“ ONIX-Element eintragen. Zum anderen kann in der ONIX-Spezifikation ein Element nicht gründlich bzw. eindeutig genug definiert sein, sodass eine nicht einheitliche Verwendung unter bestimmten Umständen erlaubt ist, wie z. B. bei `<j143>`.[139] Dieses Element ist für das Datum vorgesehen, an dem das Produkt vom Einzelhändler zum Verkauf angeboten werden kann; in Großbritannien sollte hingegen das Erscheinungsdatum („launch date“) angegeben werden:

> „The date when a new product can be placed on sale by retailers in the market served by the supplier. […] In the UK, publishers who are following the PA/BA Launch Dates Code of Practice should use this element for the Launch Date”[140].

Zuletzt können ONIX-Elemente auch versehentlich durch Metadatenmanager falsch verwendet werden.

---

[138] Vgl. *Herlinger, B.*, Zweckentfremdungen in ONIX 2.1 und 3.0, 2022.
[139] Vgl. *ebd*.
[140] *EDItEUR*, ONIX for Books: Product Information Message Product Record Format, 2006, S. 154.

Bei der Analyse der ONIX-Daten wurden jedoch auch viele andere Arten von Fehlern oder unvorteilhaften Darstellungen der Informationen entdeckt, bei denen es sich lohnt, diese genauer zu identifizieren und Schematron-Prüfregeln für fehlerfreie, konsistente ONIX-Produktinformationen zu erstellen. Generell wurden neben formalen Fehlern – also eine von der ONIX-Spezifikation abweichende Verwendung von Elementen – inhaltliche Fehler sowie Problematiken und textliche Fehler und eine ungeordnete Darstellung von Texten entdeckt. Da es sich zur Sicherung der Qualität der Metadaten und somit zum erfolgreichen und effizienten Vertrieb der Produkte lohnt, auch auf weitere Fehler hinzuweisen, wurden auch diese Problematiken zu den Schematron-Regeln aufgenommen.

Die Analyse verschiedener ONIX-Dateien erfolgte im Oxygen XML Editor hauptsächlich in drei Methoden, deren Ergebnisse als Business Rules in den nachfolgenden Kapitelabschnitten aufgezeigt werden:

1. Untersuchung von bestimmten ONIX-Elementen aufgrund von Berichten von Metadatenmanagern des Praxispartners und Reklamationen[141]
2. Lesen der ONIX-Spezifikation und Betrachtung der Elemente in ONIX-Dateien mittels XPath-Ausdrücken im XML-Editor
3. Betrachtung von ONIX-Elementen in ONIX-Dateien, die für freie Textinhalte vorgesehen sind, und Untersuchung auf textliche Fehler.

In den folgenden Abschnitten werden die aufgedeckten Fehler und Problematiken beschrieben sowie nummeriert und jeweils am Ende eines Abschnitts in einer fortlaufend nummerierten Liste als Business Rules formuliert. Business Rules beschreiben Prinzipien, die spezifisch ausgelegt werden können und die die semantische Konsistenz von Daten sicherstellen.[142] Ein Schematron-Schema ist ein geeignetes Mittel, um Business Rules technisch zu formulieren und XML-Dokumente anhand dieser Regeln zu überprüfen. Jede mögliche Bedingung, die an ein XML-Dokument gestellt werden kann, kann als Business Rule formuliert werden. Zum Abschluss des Kapitels werden die Business Rules klassifiziert.

5.1 Von Metadatenmanagern berichtete Zweckentfremdungen und Fehler

Beim Update einer ONIX-Meldung kam es zu Problemen bei Elementen, die für Biografien vorgesehen sind, die auf die im Folgenden erläuterten Gründe zurückzuführen sind. In ONIX-Meldungen besteht neben der Angabe einer Biografie zu einem Autor auch die Möglichkeit eine Gesamtbiografie aller Mitwirkenden anzugeben.

[141] Vgl. *Pufe, M.*, Wichtige ONIX-Elemente und Reklamationen, 2022.
[142] Vgl. *Lackes, R.*, Definition: Business Rule, 2018.

Eine Einzelbiografie ist für das Element <b044> vorgesehen, während die Gesamtbiografie in einem <d104>-Element beherbergt sein soll.[143] In der ONIX-Spezifikation darf im Datenelement für die Gesamtbiografie ausdrücklich nicht die Einzelbiografie eines Mitwirkenden stehen: „A note referring to all contributors to a product – NOT linked to a single contributor“[144] (s. Business Rule 1). Es könnte allerdings toleriert werden, wenn das Produkt genau einen Contributor hat und sowohl eine Einzel- als auch eine Gesamtbiografie enthalten sind. Dann sollten jedoch beide Texte genau übereinstimmen, um bei einem späteren Verarbeitungsprozess Fehler zu vermeiden (s. Business Rule 2). Zusätzlich kann bei mehreren Autoren aus deren Einzelbiografien durch einen Verarbeitungsprozess eine Gesamtbiografie erstellt werden. Falls jedoch nicht für jeden Autor eine Einzelbiografie vorhanden ist, so kann es passieren, dass im Element für die Gesamtbiografie fälschlicherweise eine Einzelbiografie steht (s. Business Rule 3).

Im Interview mit einer ONIX-Expertin wurden mehrere ONIX-Elemente erwähnt, die gelegentlich fälschlich verwendet werden. So wird im Element <b062>, das für Notizen zu Illustrationen sowie zu Karten, Tabellen usw. vorgesehen ist, fälschlicherweise die Seitenzahl angegeben, obwohl dafür das Element <b061> (<NumberOfPages>) zur Verfügung steht (s. Business Rule 4). Im Element <b058> soll laut Spezifikation eine freitextliche vollständige Beschreibung der Auflage enthalten sein, wie z. B. „3rd edition, revised with an introduction and notes“[145]. Inkorrekterweise erscheint hier gelegentlich nur die Ziffer der Auflage oder ausschließlich eine Jahreszahl. Hierfür stehen jeweils die Elemente <b057> (<EditionNumber>), <b217> (<EditionVersionNumber>), oder z. B. <b087> (<CopyrightYear>) zur Verfügung. Ebenso wurden unvollständige Beschreibungen ermittelt, wie etwa „vollständig überarbeitete Ausgabe“, oder eine falsche Verwendung des Elements, wie die Laufzeitangabe eines Audioprodukts oder die Anzahl an Bänden („Ausgabe mit 2 Bänden“) (s. Business Rule 5). Eine nicht sinngemäße Verwendung ist bei der Nutzung von Keywords zu beobachten, wenn bei einem Produkt zu viele Schlüsselwörter verwendet werden, die nicht aussagekräftig sind. So wurden zu einzelnen Buchprodukten jeweils rund 100 <b070>-Elemente genannt, die als Keyword ausgezeichnet sind (s. Business Rule 6). **Listing 5** zeigt beispielhaft drei Keywords, die in ONIX jeweils durch ein <b067>-Element mit dem Code 20 als Keyword definiert werden und in <b070>-Element angegeben werden.

---

[143] Genauer: in <d104>, wenn zuvor im selben Elementverbund in <d102> der Code 13 steht.
[144] *EDItEUR*, ONIX for Books: Codelists Issue 36, 2017, List 33, Code 13.
[145] *EDItEUR*, ONIX for Books: Product Information Message Product Record Format, 2006.

```
<product>
  <!--...-->
  <subject>
    <b067>20</b067>
    <b070>Musik</b070>
  </subject>
  <subject>
    <b067>20</b067>
    <b070>Freizeit</b070>
  </subject>
  <subject>
    <b067>20</b067>
    <b070>Hobby</b070>
  </subject>
  <!--...-->
</product>
```

Listing 5: Aufzählung von Keywords in ONIX

Grundsätzlich findet auch eine Zweckentfremdung statt, wenn Produkte aus dem Nonbooksektor wie etwa Spiele per ONIX gemeldet werden, da der Standard auf Bücher ausgerichtet ist und in ONIX 2.1 nur begrenzt für die Meldung von Spielen geeignet ist (s. Business Rule 7).[146]

Ein Geschäftspartner des Praxispartners dieser Arbeit hatte gelegentlich Probleme, wenn der Text in einem Element eine bestimmte Anzahl an Zeichen überschritt. Dies geht zwar nicht auf eine falsche Verwendung eines ONIX-Elements zurück, jedoch gibt es in der Spezifikation durchaus Empfehlungen für einige Elemente, wie etwa bei `<b203>`, für das eine maximale Länge von 300 Zeichen empfohlen wird (s. Business Rule 8).[147]

Aus den genannten Problematiken und falschen Verwendungen lassen sich folgende acht Business Rules ableiten:

1. Wenn es mehr als einem Contributor gibt, darf der Einzelbiografietext eines Contributors nicht genau mit der Gesamtbiografie aller Contributoren übereinstimmen.
2. Wenn es nur einen Contributor gibt und dieser eine Einzel- und Gesamtbiografie besitzt, müssen beide Texte genau übereinstimmen.
3. Es muss gleich viele Autoren und Einzelbiografien geben.
4. In b062 sollen keine Seitenangaben enthalten sein.
5. In b058 dürfen nicht ausschließlich Zahlzeichen vorkommen; es soll eine vollständige freitextliche Beschreibung der Auflage enthalten sein.

[146] Vgl. *Herlinger, B.*, Zweckentfremdungen in ONIX 2.1 und 3.0, 2022.
[147] Vgl. *Pufe, M.*, Wichtige ONIX-Elemente und Reklamationen, 2022.

6. Es sollen nicht zu viele Keyword-Elemente aufgeführt werden (Vorschlag: maximal 30).
7. Ein Nonbookprodukt soll nicht per ONIX gemeldet werden.
8. In b203 soll die maximale Anzahl von 300 Zeichen nicht überschritten werden.

5.2 Betrachtung der Elemente bezüglich der ONIX-Spezifikation

Für die Analyse nicht korrekt verwendeter Elemente in ONIX wurden vier ONIX-Meldungen verschiedener Verlage betrachtet. Die davon mit Abstand größte ONIX-Datei enthält etwa 425 000 Zeilen Code und liefert Metadaten zu etwa 1 350 Produkten. Es wurde die ONIX-2.1-Spezifikation studiert, woraufhin alle Elemente stichprobenartig ausgewählter `<product>`-Elemente betrachtet wurden und der Elementinhalt mit der Dokumentation abgeglichen wurde. Gab es eine Auffälligkeit, so wurden mit einer XPath-Suche (s. **Abbildung 7**), die der XML Editor Oxygen in seiner Benutzeroberfläche zur Verfügung stellt, die Elemente gesucht, was eine schnelle Überprüfung gleichnamiger Elemente ermöglichte.

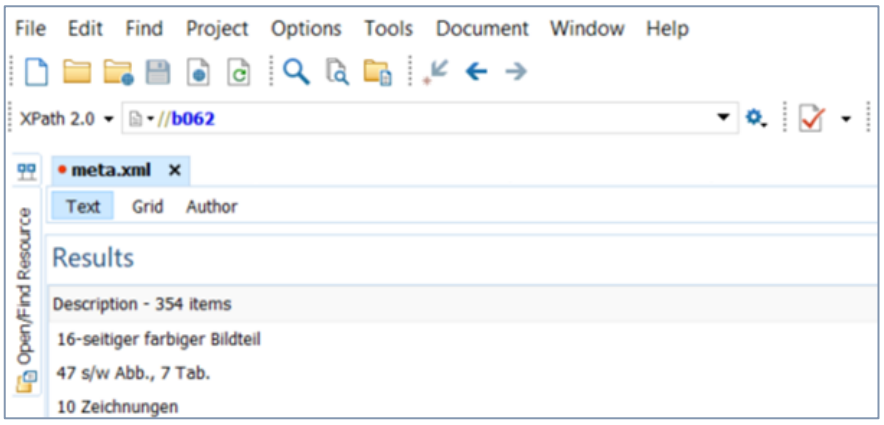

Abbildung 7: XPath-Suchleiste mit Eingabe und Resultaten im Oxygen XML Editor (Eigene Darstellung)

Im Folgenden werden Auffälligkeiten erläutert, die in den vier ONIX-Meldungen erschienen und von der Nutzung, wie es die ONIX-Spezifikation vorsieht, abweichen. Laut Spezifikation soll der Biografietext in `<b044>` immer auch den Namen des Mitwirkenden enthalten: „A biographical note in ONIX should always contain the name of the person or body concerned“[148]. Der Name des Mitwirkenden ist in `<b036>` beherbergt und es kam gelegentlich vor, dass die Namensangabe leicht unterschiedlich war, wie etwa mit fehlenden diakritischen Zeichen wie einem Akzent oder Hatschek. So ist z. B. auf der Website eines Verlages der Autorenname „Janos Klocke“ hinterlegt, während im Biografietext „Janoš Klocke ist Politikwissenschaftler […]“[149] steht. Zudem

[148] *EDItEUR*, ONIX for Books: Product Information Message Product Record Format, 2006, S. 55.
[149] https://www.campus.de/autoren/autoren-a-z/janos_klocke-8949.html

kam es vor, dass in einem Element lediglich ein Name des Autorenverbundes oder ein Pseudonym angegeben ist, während im anderen Element die realen Namen notiert sind (s. Business Rule 9). Da genannte Daten genauso in der ONIX-Meldung stehen, kann davon ausgegangen werden, dass die Informationen aus den ONIX-Metadaten auf die Website übernommen wurden. Es sollen ebenfalls keine URLs im Biografietext in <b044> notiert sein, da dies zu Problemen bei der Verarbeitung führen kann: „Some recipients of ONIX data feeds will not accept text which has embedded URLs. A contributor website link can be sent using the <Website> composite on the next page“[150] (s. Business Rule 10). Dass URLs in <b044> aufgeführt wurden, kam dennoch häufig vor.

Im Element <b062>, das für Anzahl und Typ der Illustrationen sowie weitere verwandte Notizen in Freitextform vorgesehen ist, erschienen gelegentlich als Textinhalt ausschließlich Zahlzeichen (s. Business Rule 11). Falls mit der Zahl die totale Anzahl an Illustrationen gemeint sein sollte, so gibt es dafür das Element <b125> (<NumberOfIllustrations>), das nur numerische Zeichen zulässt.

Gelegentlich war im Element <b029> eine Auflagenbeschreibung notiert, obwohl dieses Element für den Untertitel vorgesehen ist. In der Dokumentation wird die Verwendung folgendermaßen beschrieben:

> „The full text of a subtitle, if any. ‘Subtitle’ means any added words which appear with the title given in an occurrence of the <Title> composite, and which amplify and explain the title, but which are not considered to be part of the title itself“[151].

Hier wird zwar nicht eindeutig eingegrenzt, was in <b029> stehen darf; da es allerdings Elemente für die Auflagenbeschreibung und ähnliches gibt, sollten diese Elemente für die Einheitlichkeit der Metadaten auch für diese Zwecke verwendet werden (s. Business Rule 12). Teilweise waren im Untertiel-Element <b029> auch Marketinginformationen zu finden, wie etwa „[…] plus E-Book inside (ePub, mobi oder pdf)“ [152], was ebenso auf einer Website zu entdecken ist (s. Business Rule 13).

Aus den genannten Problematiken und falschen Verwendungen lassen sich folgende Business Rules ableiten:

9. Der Name des Contributors muss im Biografietext vorkommen und sollte mit dem angegebenen Namen in b036 übereinstimmen.
10. In einer Einzelbiografie (b044) sollte keine URL enthalten sein.
11. In b062 sollen nicht ausschließlich Zahlzeichen enthalten sein.
12. Im Untertitel (b029) soll keine Auflagenbeschreibung enthalten sein.
13. Im Untertitel (b029) sollen keine Marketinginformationen enthalten sein.

---

[150] *EDItEUR*, ONIX for Books: Product Information Message Product Record Format, 2006, S. 55.
[151] *Ebd.*, S. 42.
[152] https://www.campus.de/e-books/business/management-unternehmensfuehrung/praxishandbuch_produktmanagement-17513.html

5.3 Suche nach typischen textlichen Fehlern und Problematiken

Während der Analyse gelegentlich falsch verwendeter ONIX-Elemente fielen auch andere Fehler und Problematiken auf, die keine Zweckentfremdungen waren, sondern z. B. auf Kodierungsfehler zurückzuführen sind. In einem Best Practice-Leitfaden der ONIX-Anwendergruppe für den deutschsprachigen Raum wird ebenso auf Zeichenprobleme hingewiesen. Gewöhnlich werden ONIX-Meldungen mit dem Zeichensatz UTF-8 gesendet, allerdings können auch andere Zeichensätze verwendet werden, wobei wichtig ist, dass diese Information dann in der XML-Deklaration angegeben wird. Da die unterschiedlichen Informationen meist ursprünglich „aus verschiedenen Datenquellen (Datenbank, Websites, PDF …) mit unterschiedlichen Zeichensätzen stammen, kann es leicht zu Zeichensatzproblemen kommen. Gerade bei der Übernahme von Zusatztexten durch ‚Copy&Paste' können Zeichensätze vermischen. Dies führt zu Problemen in der Verarbeitung und in der Darstellung"[153]. Die fehlerhaften Texte erschienen v. a. in Elementen, die für freie Texte vorgesehen sind, wie etwa die Buchbeschreibung, Biografietexte oder Inhaltsverzeichnisse. Da Freitexte besonders oft im `<othertext>`-Verbund im Element `<d104>` vorkommen, war der Inhalt genau dieser Elemente besonders interessant und ergebnisreich. Auch das Einzelbiografie-Element `<b044>` wurde untersucht. Dafür wurden im XML-Editor mittels der XPath-Ausdrücke `//d104` sowie `//b044` die Elemente adressiert, wodurch sie effektiv prüfgelesen werden konnten. Gab es Auffälligkeiten, so wurde der entsprechende Text auf der Website des entsprechenden Verlags oder mithilfe einer Online-Suchmaschine gesucht und bei Ergebnissen mit den Metadaten verglichen. Es zeigte sich, dass die Auffälligkeiten, die online zu sehen waren, im Großteil mit den Metadaten übereinstimmten.

Generell gibt es dabei drei verschiedene Arten von Problematiken: inhaltliche Problematiken, sprachliche Defizite wie Rechtschreibfehler und Zeichenfehler in Form von Kodierungsfehlern. Allen ist gemeinsam, dass sie sich mithilfe von regulären Ausdrücken aufspüren lassen. Im Folgenden werden die verschiedenen Auffälligkeiten erläutert.

In Einzelbiografietexten erschien Text, der zeitabhängig ist und somit veralten kann, wenn er nicht stetig aktualisiert wird. Gemeint ist damit Text im Muster von „Seit/seit … Jahren", wie z. B. „[…] leitet seit 5 Jahren den Content-Bereich […]"[154] (s. Business Rule 14).

---

[153] *ONIX Anwender-Gruppe für den deutschsprachigen Raum*, Best Practices ONIX for Books (Version 2.1), 2010, S. 10.
[154] https://www.campus.de/autoren/autoren-a-z/ben_hughes-8402.html

Es fielen fehlerhafte Worttrennungen auf, wie etwa „be-nutzen“ oder „ostasiati-sche“[155] (s. Business Rule 15). Rechtschreibfehler, bei welchen fälschlicherweise drei identische Buchstaben aufeinander folgten, traten auch gelegentlich auf, wie z. B. bei „Influencerinnnen“[156] (s. Business Rule 16).

Zwischen zwei Sätzen fehlten nach dem Satzzeichen gelegentlich Leerzeichen, wie etwa bei „[…] Erfolgsstrategien von Deutschlands Trainerelite in einem Buch:Wer im Job weiterkommen und zufriedener leben will […]“[157] oder bei „[d]er französische Aristokrat entdeckt in ihr die exklusive Gesellschaftsform (´etat social) derModerne.Galt sie den Denkern des 18. Jahrhunderts noch als eine Staatsform[…]“[158] (s. Business Rule 17). Dabei zeigt sich beim letzten Beispiel, dass fehlende Leerzeichen auch Hinweise auf weitere textliche Fehler im entsprechenden Freitext geben können: So sollte es korrekt „état social“ und „der Moderne“ heißen.

Es fielen Inhaltsverzeichnisse – teilweise auch mit Seitenzahlen – auf, die im XML-Editor unstrukturiert erscheinen. Beim Vergleich mit entsprechenden Einträgen auf Websites erscheinen die Inhaltsverzeichnisangaben auch ungeordnet als Fließtext und ohne Umbrüche, wie etwa „Inhalt Vorwort der Herausgeber Neue Erfolgsfakten 7 Dieter Brandes Einfach managen 10 Erfolgreiche Unternehmensführung mit Vertrauen und Kontrolle […]“[159], was beim Anblick potenzieller Käufer diese vom Kauf abhalten könnte. In diesem Fall lohnt es sich, gezielt nach Freitext-Elementen zu suchen, die *Table of Contents* (Element `<d102>` mit Code `04`) im Default-Textformat (Element `<d103>` mit Code `06`) angeben (s. Business Rule 18). Laut ONIX-Codelist ist es dennoch erlaubt, unstrukturierten Text für TOCs zu verwenden:

> „Used for a table of contents sent as a single text field, which may or may not carry structure expressed through HTML etc. Alternatively, a fully structured table of contents may be sent by using the <ContentItem> composite.“ [160]

Allerdings zeigt sich dabei, dass der unstrukturierte Text trotzdem veröffentlicht werden kann und es bessere Alternativen gibt, ein Inhaltsverzeichnis gut leserlich im WWW anzugeben. Wie bereits in **Abschnit 4.2.2** erwähnt, ist es in bestimmten ONIX-Elementen erlaubt, XHTML zu nutzen. Im Onlinedokument „Application Note: Embedding HTML markup in ONIX 3.0 data elements“[161] gibt EDItEUR Hinweise zur Nutzung von Auszeichnungen innerhalb von ONIX-Tags. Darin wird auch darauf hingewiesen, dass bei inkonsistenter Verwendung von Auszeichnungen bestimmte Texte monolithisch und als

---

[155] https://www.campus.de/autoren/autoren-a-z/werner_schwanfelder-910.html
[156] https://www.campus.de/autoren/autoren-a-z/annahita_esmailzadeh-8967.html
[157] https://www.lehmanns.de/shop/sachbuch-ratgeber/22318029-9783593413686-die-erfolgsmacher-ii-von-den-besten-profitieren
[158] https://www.amazon.de/Alexis-Tocqueville-Campus-Einf%C3%BChrungen-Karlfriedrich-ebook/dp/B004WN7VB2
[159] https://www.eurobuch.com/buch/isbn/9783593413686.html
[160] *EDItEUR*, ONIX for Books: Codeslists Issue 59, 2022, List 33, Code 06.
[161] *EDItEUR*, ONIX for Books: Product Information Message, 2021.

„unschöner Block von unformatiertem Text“[162] dargestellt werden können. Diese Problematik kann für weitere Regeln auch als Grundlage dafür genommen werden, um zu prüfen, ob XHTML-Auszeichnungen wie z. B. `<br/>` als Elementinhalt vorkommen, ohne dass der Code `05` für *XHTML*[163] angegeben ist.

Zwischen Zahlzeichen im Tausenderbereich traten einzelne Fragezeichen auf, was vermutlich auf Kodierungsfehler zurückzuführen ist, wie bei „200?000“[164] (s. Business Rule 19). Ebenso folgten gelegentlich Fragezeichen nach Vokalen, die eigentlich Umlaute sein sollten, wie etwa „religio?s“, „gegenu?ber“ oder „fu?r“ (s. Business Rule 20).

Mitten im Text erschien auch – semantisch und syntaktisch inkorrekt – ein Fragezeichen, das von zwei Leerzeichen umgeben war, wie z. B. in „[…] sondern eine sinnvolle und wichtige Investition in die Jobzufriedenheit ? und Ihren Erfolg!“[165] (s. Business Rule 21). Weitere Auffälligkeiten mit inkorrekt auftretenden Fragezeichen zwischen anderen Zeichen wurden auch anderen Stellen entdeckt, wo vermutlich geschützte Leerzeichen oder andere Sonderzeichen wie Sternchen stehen sollten. Beispiele sind „u.?a.“ , „St.?Galler Business School“[166], „Ay?a Polat“, „Privatpersonen als Spen?der?Innen“, „113?ff“ oder „S.?41“. Damit diese fehlerhaften Darstellungen sich nicht mit den im Abschnitt zuvor genannten eindeutiger bestimmbaren Fehlern überschneiden, werden hierfür weitere Business Rules formuliert (s. Business Rules 22-24).

Es ergeben sich folgende Business Rules für textliche und inhaltliche Problematiken:

14. Es soll kein Text vorhanden sein, der von der Zeit des Verfassens des Textes abhängt und damit veralten kann. Insbesondere gilt dies für Text nach dem Muster „seit … Jahren“ oder „Seit … Jahren“.
15. Es sollen keine fehlerhaft mit Bindestrich getrennten Wörter erscheinen.
16. Rechtschreibfehler mit drei identischen Buchstaben hintereinander sollen nicht auftreten.
17. Zwischen dem Satzzeichen am Ende eines Satzes und dem Beginn eines neuen Satzes soll kein Leerzeichen fehlen.
18. Ein Inhaltsverzeichnis sollte nicht im Default-Textformat erscheinen.
19. Zwischen Zahlzeichen soll sich kein Fragezeichen befinden.
20. Zwischen einem Vokal und einem Buchstaben soll sich kein Fragezeichen befinden.
21. Ein Fragezeichen soll nicht von zwei Weißraumzeichen umschlossen sein.

---

[162] *EDItEUR*, ONIX for Books: Product Information Message, 2021, S. 2.
[163] Vgl. *EDItEUR*, ONIX for Books: Codelists Issue 36, 2017, List 34, Code 05.
[164] https://buchundmedien.com/shop/item/9783593424859/limbi-von-werner-tiki-kustenmacher-e-book-epub
[165] https://www.legimi.de/e-book-kenn-ich-alles-marco-von-munchhausen,b427187.html
[166] https://www.lehmanns.de/shop/wirtschaft/50732657-9783593511719-das-konzept-integriertes-management

22. Nach einem Zeichen (außer einem Vokal) und darauffolgendem Fragezeichen soll nicht direkt ein kleiner Buchstabe folgen.
23. Nach Wortzeichen mit darauffolgendem Punkt soll direkt dahinter kein Fragezeichen erscheinen, wenn darauf ein Großbuchstabe folgt.
24. Nach einem Zeichen, außer einer Ziffer, soll direkt dahinter kein Fragezeichen erscheinen, wenn darauf eine Ziffer folgt.

5.4 Klassifizierung der Business Rules

Für die Erstellung eines übersichtlichen und nutzerfreundlichen Schematron-Schemas bietet es sich an, die Prüfregeln anhand gemeinsamer Merkmale in verschiedene Kategorien einzuteilen, die später zu sog. Phasen werden. In einem `<phase>`-Element wird eine Klasse von Patterns festgelegt, die bei einer Schematron-Validierung aktiviert werden kann, während andere Patterns inaktiv bleiben. Jede Phase erhält auch einen Namen in Form einer ID, um bei der Validierung die Klasse der Patterns zu aktivieren.

Für die Erstellung des Schematron-Schemas werden die erstellten Business Rules zunächst nach ihrer Fehlerart bzw. ihrem Prüfzweck klassifiziert. Dabei ergeben sich nach einer Analyse die vier verschiedenen Kategorien *falsche Verwendung, Biografiefehler, textliche Fehler* und *Empfehlungen*. Die Einordnung der Regeln in diese Kategorien stellen auch vier verschiedene Phasengruppen dar.

Zusätzlich wurde für jede Business Rule der Schweregrad charakterisiert. Für diesen wurde jede Fehlerproblematik danach beurteilt, ob der Gebrauch eines Elements von der Definition in der ONIX-Dokumentation abweicht, ob sie Problematik in der Vergangenheit zu Verarbeitungsproblemen führte und inwiefern sie eine ordentliche Darstellung und Lesbarkeit beeinflusst, sobald die Daten automatisiert übernommen werden und z. B. für potenzielle Händler oder Endverbraucher zu lesen sind. Die Schweregrade können in einem `role`-Attribut definiert werden. Wie `<phase>` und `role="..."` umzusetzen sind, wird in **Kapitelabschnitt 6.1** behandelt.

Im Folgenden sind die nach Klasse bzw. Phasen eingeordneten Business Rules zu sehen. Die Nummer entspricht der Aufzählung in den **Abschnitten 5.1 bis 5.3** und der Regelname ist der Bezeichner des Patterns mittels einer ID und somit Namensgeber für die Referenzierung innerhalb der Phasen. Die verschiedenen Schweregrade sind *Fehler*, *Warnung* und *Hinweis*.

Falsche Verwendung

In **Tabelle 9** folgen die Business Rules, die falsche bzw. zweckfremde Verwendung von ONIX-Elementen aufspüren. Der ID-Name der Phase, die diese Business Rules in Form von Schematron-Prüfregeln enthalten wird, lautet „FalscheVerwendung“.

Nr.	Business Rule	Regelname	Schweregrad
4	In b062 sollen keine Seitenangaben enthalten sein.	**AbbildungsnotizSeiten**	Warnung
5	In b058 dürfen nicht ausschließlich Zahlzeichen vorkommen; es soll eine vollständige freitextliche Beschreibung der Auflage enthalten sein.	**EditionsbeschreibungFehler**	Fehler
7	Ein Nonbookprodukt soll nicht per ONIX gemeldet werden.	**Nonbookmeldung**	Warnung, ggf. Hinweis
11	In b062 sollen nicht ausschließlich Zahlzeichen enthalten sein.	**AbbildungsnotizZiffer**	Fehler
12	Im Untertitel (b029) soll keine Auflagenbeschreibung enthalten sein.	**UntertitelAuflage**	Fehler
13	Im Untertitel (b029) sollen keine Marketinginformationen enthalten sein.	**UntertitelErweiterung**	Warnung[167]

Tabelle 9: Klassifizierung der Business Rules zur Kategorie „Falsche Verwendung“

Biografiefehler

Obwohl bei einigen Business Rules, die die Biografien der Contributoren betreffen, gleichzeitig eine falsche Verwendung der Elemente vorliegt, wurde entschieden, die Biografiefehler als eigene Kategorie zu definieren, da diese Elemente zu einem wichtigen Bereich gehören, der oft extrahiert wird und bereits aus verschiedenen Gründen bei inkorrekter Verwendung zu Problemen führte.[168] **Tabelle 10** zählt die Business Rules auf, die Fehler und Problematiken melden, die die Einzel- und/oder Gesamtbiografie-Metadaten betreffen. Der ID-Name dieser Phase lautet „BiografieFehler“.

[167] Es wurde „nur“ der Schweregrad Warnung ausgewählt, da es sich möglicherweise um eine bewusste Entscheidung zur Verkaufsstrategie handelt.
[168] Vgl. *Pufe, M.*, Wichtige ONIX-Elemente und Reklamationen, 2022.

Nr.	Business Rule	Regelname	Schweregrad
1	Wenn es mehr als einen Contributor gibt, darf der Einzelbiografietext eines Contributors nicht genau mit der Gesamtbiografie aller Contributoren übereinstimmen.	**ContributorGesamtbio**	Fehler
2	Wenn es nur einen Contributor gibt und dieser eine Einzel- und Gesamtbiografie besitzt, müssen beide Texte genau übereinstimmen.	**EinContributorBios**	Fehler, da es zu Problemen führen kann
3	Es sollte gleich viele Autoren und Einzelbiografien geben.	**AutorenBios**	Warnung, da es nur u. U. zu Problemen führen kann
9	Der Name des Contributors muss im Biografietext vorkommen und sollte mit dem angegebenen Namen in b036 übereinstimmen.	**BioName**	Fehler
10	In einer Einzelbiografie (b044) sollte keine URL enthalten sein.	**BioURL**	Fehler, da es zu Problemen führen kann

Tabelle 10: Klassifizierung der Business Rules zur Kategorie „Biografiefehler“

Textliche Fehler

Die in **Tabelle 11** aufgezählten Business Rules spüren Zeichenfehler auf. Die Fehler **Nr. 21-24** werden in einem `<pattern>` mit der ID „SonstigeKodierungsfehler“ zusammengefasst, da die Kodierungsfehler ähnlich aussehen, und sind zur Verständlichkeit mit den Buchstaben *A-D* kommentiert. Weiterhin sind hier Business Rules eingeordnet, die eine mögliche "unordentliche" Darstellung von Texten durch bestimmte Rechtschreibfehler aufspüren. Der ID-Name dieser Phase lautet „TextFehler“.

Nr.	Business Rule	Regelname	Schweregrad
15	Es sollen keine fehlerhaft mit Bindestrich getrennten Wörter erscheinen.	**Bindestrich**	Hinweis
16	Rechtschreibfehler mit drei identischen Buchstaben hintereinander sollen nicht auftreten.	**DreiBuchstaben**	Hinweis
17	Zwischen dem Satzzeichen am Ende eines Satzes und dem Beginn eines neuen Satzes soll kein Leerzeichen fehlen.	**FehlendesSpace**	Warnung
19	Zwischen Zahlzeichen soll sich kein Fragezeichen befinden.	**ZahlFehler**	Warnung
20	Zwischen einem Vokal und einem Buchstaben soll sich kein Fragezeichen befinden.	**UmlautFehler**	Warnung
21	Ein Fragezeichen soll nicht von zwei Weißraumzeichen umschlossen sein.	**SonstigeKodierungsfehler (A)**	Warnung
22	Nach einem Zeichen (außer einem Vokal) und darauffolgendem Fragezeichen soll nicht direkt ein kleiner Buchstabe folgen.	**SonstigeKodierungsfehler (B)**	Warnung
23	Nach Wortzeichen mit darauffolgendem Punkt soll direkt dahinter kein Fragezeichen erscheinen, wenn darauf ein Großbuchstabe folgt.	**SonstigeKodierungsfehler (C)**	Warnung
24	Nach einem Zeichen, außer einer Ziffer, soll direkt dahinter kein Fragezeichen erscheinen, wenn darauf eine Ziffer folgt.	**SonstigeKodierungsfehler (D)**	Warnung

Tabelle 11: Klassifizierung der Business Rules zur Kategorie „Textliche Fehler“

Empfehlungen

In **Tabelle 12** folgen die Business Rules für die Klasse „Empfehlungen“, die Problematiken identifizieren, die zu unschöner Darstellung des Textinhalts führen können oder ineffizient wie auch zeitsensitiv sind. Der Name der `<phase>`-ID lautet für diese Klasse „Empfehlungen“.

Nr.	Business Rule	Regelname	Schweregrad
6	Es sollen nicht zu viele Keyword-Elemente aufgeführt werden (Vorschlag: maximal 30).	**KeywordInflation**	Hinweis
8	In b203 soll die maximale Anzahl von 300 Zeichen nicht überschritten werden.	**Zeichenlaenge**	Hinweis
14	Es soll kein Text vorhanden sein, der auf die Zeit des Verfassens des Textes abhängig ist und damit veralten kann, nach dem Muster „seit … Jahren“ oder „Seit … Jahren“.	**SeitJahren**	Warnung
18	Ein Inhaltsverzeichnis sollte nicht im Default-Textformat erscheinen.	**TOCDefaultTextFormat**	Hinweis

Tabelle 12: Klassifizierung der Business Rules zur Kategorie „Empfehlungen“

6 Erstellung aussagekräftiger Schematron-Regeln für ONIX 2.1

Dieses Kapitel behandelt die Erstellung der Schematron-Regeln auf Grundlage der erstellten Business Rules. Dazu wird betrachtet, welche Möglichkeiten genutzt werden, um das Schematron-Schema benutzerfreundlich zu gestalten. Anschließend wird die Erstellung jeder Regel mit einer reduzierten Darstellung der Patterns erläutert. Im letzten Abschnitt wird kurz darauf eingegangen, wie mithilfe von XSL-Transformationen ein optisch ansprechender Schematron-Fehlerreport unabhängig von der Entwicklungsumgebung erzeugt werden kann.

Im **Anhang A** sind alle Prüfregeln in der gleichen Reihenfolge wie im Folgenden erläutert dargestellt, inkl. den Angaben der Schweregrade und den vollständigen XSLT-Instruktionen.

6.1 Nutzerfreundliche Formulierung der Regeln in Schematron

Im Folgenden werden die Mittel beschrieben, mit welchen die Business Rules zu benutzerfreundlichen Schematron-Prüfregeln überführt werden. Insgesamt soll das Schematron-Schema von externen Metadatenpflegern verstanden, angewandt und bei Wunsch individuell angepasst werden können. Auch die Fehlermeldungen sollen für die Zielgruppe gut verständlich sein und ggf. Hinweise geben, um eine manuelle Korrektur in die Wege zu leiten.

6.1.1 Erstellung von Phasen

Standardmäßig werden bei einer Validierung gegen ein Schematron-Schema alle Patterns aktiviert. In der Implementierung im Oxygen XML Editor wird auch als Auswahlmöglichkeit „#ALL“ angezeigt, was einer Validierung ohne definierte Phasen – also der Standardeinstellung – entspricht. Wie in **Abschnitt 5.4** erwähnt, erlaubt es Schematron, verschiedene Patterns in sog. Phasen zu gruppieren, welche bei einer Validierung entweder aktiviert werden oder deaktiviert bleiben können. Eine Phase wird mit dem Element `<phase>` definiert und benötigt ein `id`-Attribut, damit die Schematron-Implementierung bei der Auswahl der Phase das entsprechende Pattern mit derselben ID referenzieren kann. Da die Phase einen bestimmten Prüfzweck erfüllen soll, beträgt der `id`-Attributwert im Schematron-Schema die Kurzbeschreibung des Prüfzwecks, wie etwa `id="FalscheVerwendung"`. Diese ID wird zu Beginn der Validierung als Auswahl angezeigt.

Innerhalb des <phase>-Elements befinden sich <active>-Elemente, die die Sammlung der <pattern>-Elemente bildet. Ein <active>-Element referenziert mit einem <pattern>-Attribut genau ein Pattern. Dies setzt voraus, dass jedes Pattern, das referenziert werden soll, eine ID besitzt, da jedes Pattern über seine ID referenziert wird.[169] Auch hier wird jeweils eine Kurzbeschreibung des Business-Rule-Zwecks als ID-Wert vergeben (z. B. id="ContributorGesamtbio").

Damit der Nutzer des Schematron-Schemas ein einzelnes Pattern für die Validierung aktivieren kann, wird eine Einzeltest-Phase erstellt, die in **Listing 6** zu sehen ist. Um ein einzelnes Pattern auswählen zu können, muss der Nutzer nur den Attributwert des <pattern>-Attributs in der Schematron-Datei entsprechend ändern und speichern.

```
  <phase id="EinzelTest">
    <active pattern="ContributorGesamtbio"/><!--Für Anpassung dem
Attributwert die ID des entspr. Patterns geben-->
  </phase>
```

Listing 6: Phase für die Aktivierung eines einzelnen Patterns

Im **Anhang B** sind die <phase>-Elemente für das finale Schematron-Schema zu Beginn bei **[1]** zu sehen.

6.1.2 Hierarchisierung der Fehlermeldungen

Da ONIX-Dokumente i. d. R. große Instanzen sind und eine Vielzahl von Schematron-Prüfregeln vorliegt, die nicht alle gleichwertig sind, ist die Untergliederung der Fehler in eine Fehlerhierarche sinnvoll. Mithilfe des role-Attributs, das in <report>, <assert> oder <rule> angegeben werden kann, können den Tests unterschiedliche Wertigkeiten zugewiesen werden. Sowohl die Auswirkung der angegebenen Wertigkeiten als auch die Bezeichnungen sind nicht im Schematron-Standard definiert, sondern werden von der unterstützenden Implementierung vorgegeben. Da für diese Arbeit mit dem Oxygen XML Editor gearbeitet wurde, werden die Bezeichnungen dieser Implementierung kurz vorgestellt:

- *fatal* für kritische Fehler
- *error* für Fehler
- *warning* bzw. *warn* für Warnungen
- *information* bzw. *info* für Informationen bzw. Hinweise.

Wird die Wertigkeit im role-Attribut in einem <rule>-Element angegeben, so wird sie auf alle Tests dieser Regel vererbt, denen kein anderer role-Wert zugeordnet wurde. Wird einer Regel kein role-Attribut zugewiesen, so behandelt Oxygen XML Editor den Test standardmäßig als Fehler bzw. „error“. Auftretende Fehlerreports nach einer Validierung gegen das Schematron-Schema werden ihrer Wertigkeit entsprechend

[169] Vgl. *Hedler, M./Montero Pineda, M./Kutscherauer, N.*, Schematron, 2011, S. 72–74.

dargestellt: Die fehlerhaften Kontextelemente werden verschiedenfarbig unterschlängelt und im Resultate-Ausgabefenster erhalten die Reporttexte unterschiedliche Symbole, wie in **Abbildung 8** zu sehen ist.

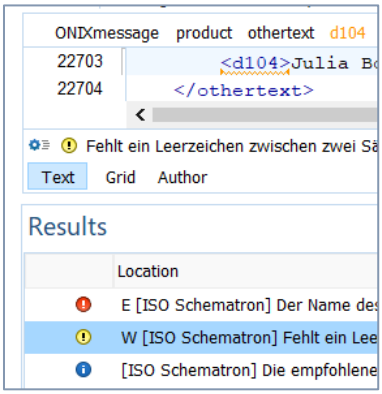

Abbildung 8: Symbole bei Fehlerreports unterschiedlicher Schweregrade im Oxygen XML Editor (Eigene Darstellung)

In den **Tabellen 9 bis 12** in **Abschnitt 5.4** wurden den Schematron-Prüfmeldungen Schweregrade zugeordnet, welche als die entsprechenden Wertigkeiten in die `role`-Attribute in die Prüfregeln übernommen werden. Da kritische Fehler bei ONIX-Dateien durch die dringend empfohlene Verwendung eines von EDItEUR zur Verfügung gestellten Schema-Dokumentes praktisch auffallen sollen, wird dieser Typ des Schweregrades nicht benötigt.

Die Reihenfolge der Patterns im Schematron-Schema kann auch für eine nutzerfreundliche Anzeige der Reports eine Rolle spielen. Nach einer Validierung gegen das Schematron-Schema erscheinen die Fehlermeldungen i. d. R. im Output (in SVRL bzw. im Resultate-Fenster im Oxygen XML Editor) in der Reihenfolge, wie die Patterns im Schematron-Dokument aufgelistet sind. Die Reihenfolge der `<active>`-Elemente in den Phasen haben darauf keinen Einfluss. Aus diesem Grund werden nach der Erstellung aller Regeln die Patterns im Dokument in einer sinnvollen Reihenfolge sortiert. Die Idee lautet, die Patterns, die kategorisch zusammengehören – also wie sie auch in den Phasen sortiert sind – zunächst hintereinander anzuordnen. Dabei sollen die Patterns mit „error“-Wert jeweils zu Beginn kommen und die Patterns mit „warning“-Wert folgen. Am Ende des Schematron-Dokumentes werden alle Patterns mit „information“-Wertigkeit angeordnet. So wird sichergestellt, dass im Output die wichtigeren Reports zuerst angezeigt werden.

6.1.3 Sprachliche Gestaltung der Fehlermeldungstexte

Für eine nutzerfreundliche Formulierung der Schematron-Prüfregeln ist es wichtig, die Zielgruppe für das Schema zu definieren sowie die Erfahrungen mit ONIX und Erwartungen, die sie an eine Schematron-Validierung hat, zu analysieren. In diesem Abschnitt geht es vor allem um die sprachliche Formulierung der Fehlermeldungstexte, wozu auch die Umstände der Verfügbarkeit des Schematron-Schemas betrachtet werden müssen.

Die Zielgruppe sind zunächst Routinebenutzer von ONIX sowie Experten aus dem deutschsprachigen Raum, die sich gut mit dem Datenaustauschformat auskennen.[170] Dazu zählen Metadatenmanager und Mitarbeiter entlang der gesamten Buchbranche, Verlagsdienstleister und Entwickler, welchen allen gemeinsam ist, dass sie XML-Kenntnisse und Erfahrung mit ONIX haben. Vermutlich traten schon Probleme mit fehlerhafter Verwendung von ONIX-Elementen auf, sodass sie sich wünschen, dass eine Validierung von ONIX-Meldungen verschiedene Problematiken aufdeckt und Hinweise gibt, womit spätere Probleme verhindert werden können. Metadatenmanager in Verlagen arbeiten oft mit einer Verlagssoftware wie z. B. *Klopotek*, aus welcher die ONIX-Dateien erzeugt werden.[171] Perspektivisch könnte das Schematron-Schema mit `<diagnostic>`-Texten angereichert werden, die das Problem und die Lösung für spezifische Nutzergruppen (z. B. Klopotek-Nutzer) erläutern. Das Schematron-Element `<diagnostic>` ist Kind des `<diagnostics>`-Elements, das einen Container für Lösungsvorschläge bildet.[172] Durch ID-Konventionen beim `<diagnostic>`-Element könnten spezifische Nutzergruppen für sie als unrelevant eingeordnete Lösungsvorschläge ignorieren.

Da das Schematron-Schema open source verfügbar[173] ist, ist die Zugänglichkeit für die Benutzergruppe gegeben. Falls beim Benutzer keine Erfahrung mit Schematron vorliegt, so kann bei Interesse an der Nutzung des Schemas und bei Fragen der bereitstellende Praxispartner kontaktiert werden.

Die Erwartungen der Benutzergruppe werden mit Blick auf ISO 9241-110 „Grundsätze der Dialoggestaltung“ von den Eigenschaften und Anforderungen beeinflusst, die das Datenaustauschformat ONIX stellt. Eine Herausforderung stellt dabei dar, dass die Short- und Reference-Tag-Syntax existieren und diese jeweils in Version 2.1 und 3.0. Um alle Voraussetzungen zu schaffen, jede dieser ONIX-Optionen gegen das Schematron-Schema validieren zu können, bieten sich im Vorfeld XSL-Transformationen an, worauf kurz in **Abschnitt 6.3** eingegangen wird. Diese helfen auch, die Schematron-Validierung

---

[170] Vgl. *Herczeg, M.*, Software-Ergonomie, 2018, S. 127.
[171] Vgl. *Pufe, M.*, Wichtige ONIX-Elemente und Reklamationen, 2022.
[172] Vgl. *Hedler, M./Montero Pineda, M./Kutscherauer, N.*, Schematron, 2011, S. 126.
[173] https://github.com/transpect/schema-onix

unabhängig vom XML-Editor durchzuführen und dabei einen HTML-Report mit den Fehlermeldungen und Fehlerstellen zu erhalten.

Ein Kriterium für die Benutzerfreundlichkeit des Schematron-Schemas ist die Erwartungskonformität: Es sollte das Vokabular verwendet werden, das dem Benutzer bei der Ausführung der Aufgabe oder aufgrund seiner Kenntnisse und Erfahrung vertraut ist. Da die ONIX-Spezifikation englischsprachig ist und auch die ONIX-Reference-Tags auf Englisch formuliert sind, werden in den Fehlermeldungstexten die englischen Schlüsselbegriffe im deutschen Fehlermeldungstext verwendet. So wird beispielsweise, um Missverständnisse zu vermeiden, das Wort „Contributor“ anstelle von „Mitwirkender“ genutzt, um darauf hinzuweisen, dass es um eine angegebene Person im `<contributor>`-Verbund geht.

In Bezug auf die Aufgabenangemessenheit sollten dem Benutzer solche Informationen angezeigt werden, die im Zusammenhang mit der erfolgreichen Erledigung der Aufgabe stehen. Die Aufgaben sind hier, eine ONIX-Datei gegen das Schematron-Schema zu validieren, auftretende Fehlermeldungen zu begutachten und die Problematiken zu beheben. Da ordentliche Metadaten positive Einflüsse auf die Distribution haben, sollen auch Hinweise einen Anreiz dazu geben, die Buchmetadaten zu pflegen. Informationen für die effiziente, erfolgreiche Erledigung sind neben der Einteilung des Schweregrades und des entsprechenden Kontextknotens auch die Angabe der Kontextelemente im Text als Short-Tag und Reference-Tag. Auch die Ausgabe von Zeichenketten mittels XSLT unterstützt die effiziente Erledigung der Aufgaben. Wenn eine Lösung für einen Fehler, eine Warnung oder für einen Hinweis es wahrscheinlich nötig hätte, die ONIX-Spezifikation zur Hand zu nehmen, so wird ein Vorschlag zur schnelleren Lösung mitgegeben. Auch bei Hinweisen lohnt es sich, für das Sicherstellen der Konsistenz der Daten einen kurzen Lösungsvorschlag mitzugeben. Die Informationsdarstellung in den Fehlermeldungen ist strukturiert organisiert, sodass sie möglichst selbstverständlich ist und als natürlich empfunden wird. So wird, wenn es sich nur *möglicherweise* um einen Fehler handelt, auch das Fehlerreport eingebunden. Außerdem folgen Lösungsvorschläge immer nach der Bezeichnung „Vorschlag:“, und wenn durch XSLT-Ausgaben Zeichenketten wiedergegeben werden, erscheinen diese immer nach der Bezeichnung „Fundstelle(n):“ (s. **Anhang A [1]**). Auf eine kurze Länge der Fehlermeldungen wurde geachtet, weshalb auch der informelle Imperativ genutzt wurde.[174]

Die Orientierung innerhalb des Schematron-Schemas wird erleichtert, indem, wie bereits erwähnt, die ONIX-Tags sowie Zeichenketten aus gemeldeten Knoten wiedergegeben werden. Bei den String-Ausgaben mittels XSLT können die Benutzer eine

[174] Vgl. *Herczeg, M.*, Software-Ergonomie, 2018, S. 220–227.

Suchen-Funktion des XML-Editors verwenden, in welchem das ONIX-Dokument geöffnet ist, um die Fehlerstelle schnell zu finden.

Die Individualisierbarkeit des Schematron-Schemas auf die eigenen Bedürfnisse des Benutzers ist gegeben, indem das Schema in einem XML-Editor bearbeitet werden kann, während eine Original-Schematron-Version open source bereitsteht. Zudem wird innerhalb des Schemas eine Orientierung gegeben, indem jedem Pattern eine ID zugewiesen wurde, die in der Benennung grob die Regel beschreibt. Durch Kommentare innerhalb des Schemas (s. **Anhang B [2]**) wird ein grober Überblick gegeben, falls das Schema z. B. durch einfache Änderungen angepasst werden kann.[175]

6.1.4 Dynamische Ausgabe von Text mittels XSLT

XSLT-Aufrufe bzw. -Ausgaben sind nicht zwingend notwendig, um fehlerhafte Kontextelemente des ONIX-Quelldokuments anzuzeigen. Zwar unterstützt der Oxygen XML Editor den Nutzer, indem er das fehlerhafte Start-Tag des Kontextelements farblich unterschlängelt, allerdings wird nicht die fehlerhafte Stelle bzw. Zeichenkette innerhalb eines Textknotens angezeigt. Jedoch wird gerade bei längeren Texten auf den ersten Blick nicht ersichtlich, wo genau der Fehler zu finden ist, um ihn schnell beheben zu können. Gerade bei größeren XML-Dokumenten kann das eine Schwierigkeit darstellen. Um die Zeichenkette, die den Fehlerreport ausgelöst hat, in der Fehlermeldung wiederzugeben, können innerhalb von Schematron XSLT-Instruktionen verwendet werden. Die Bedeutung der wichtigsten XSLT-Elemente für die Arbeit können in **Abschnitt 2.2.2** nachgelesen werden.

Im Schematron-Schema dieser Arbeit existieren Prüfungen, bei denen das Vorkommen von Zeichenketten gemeldet werden soll, die mittels regulärer Ausdrücke gefunden werden. Diese Zeichenketten und ggf. etwas Kontext sollen aus dem unter Umständen langen Textinhalt des Elements herausgezogen werden und kommasepariert in den Meldungstext einfließen. Um diese Textanalysen durchzuführen, könnte die XPath-3.1-Funktion `analyze-string()` verwendet und das Ergebnis in `<let>`-Variablen gespeichert werden. Möglicherweise würde das aber nicht mit den XSLT-2.0-basierten Stylesheets funktionieren, die aus einem Schematron-Schema XSLT erzeugen. Eine Alternative dazu, die auch mit XSLT 2.0 bzw. XPath 2.0 funktioniert, ist es, XSLT-Instruktionen in Schematron einzubetten. Diese Möglichkeit besteht im Oxygen XML Editor mit dem Setzen der *allow-foreign-Option.*[176] Deshalb werden, anstatt mit `<let>`, Variablen mit `<xsl:variable>` definiert (s. **Anhang A [1]**), wobei die Deklaration stets ein `<xsl:analyze-string>` enthält (s. **Anhang A [2]**).

---

[175] Vgl. *Herczeg, M.*, Software-Ergonomie, 2018, S. 242.

[176] „Allow foreign elements“ lautet eine Einstellung im Oxygen XML Editor, die es in Schematron gestattet, XSLT zu verwenden.

Teilweise gibt es Ausnahme-Strings wie z. B. „e.V.“ oder „Ph.D.“, die zwar auch auf generische reguläre Ausdrücke passen (s. **Abschnitt 5.3** Business Rule 17), die aber keine Fehlermeldung hervorrufen sollen. Deswegen wird in `<xsl:matching-substring>` mit `<xsl:if>` (s. **Anhang A [3]**) zusätzlich geprüft, ob es sich nicht um eine solche Ausnahme handelt. Nur wenn diese `<xsl:variable>` Textstellen enthält (s. `exists($VPunkt)` in **Anhang A** bei **[4]**), wird dann der Meldungstext ausgegeben. Aus Performancegründen wird jedoch nicht immer in jedem Kontext, dessen Textinhalt auf einen regulären Ausdruck matcht, sofort die Liste der Textstellen mit `<xsl:analyze-string>` erzeugt. Für einige Elemente bzw. reguläre Ausdrücke hat es sich als wesentlich effizienter erwiesen, zunächst nur zu testen, ob ein regulärer Ausdruck anschlägt, wie z. B. bei

```
<report test="matches(., $ZahlFehler-Regex)">,
```

um anschließend innerhalb des `<report>`-Elements (s. **Anhang A [5]**) mit `<xsl:analyze-string>` die einzelnen Fundstellen zu ermitteln. Der Nachteil bei diesem Vorgehen ist, dass keine Ausnahmen berücksichtigt werden können, bzw. dass, falls es Ausnahmen gibt, der `<report>` trotzdem zunächst anschlägt, es jedoch keine Textstellen gemeldet werden.

Für die regulären Ausdrücke, bei denen es Ausnahmen gibt, werden die Fundstellenlisten berechnet, sobald die `<rule>` anschlägt. Im finalen Schematron-Schema wird das nur bei der Prüfregel *FehlendesSpace* angewandt. Die sechs anderen `<xsl:analyze-string>`-Instruktionen konnten performance-optimiert ins `<report>`-Element verschoben werden und werden nur berechnet, wenn `test="matches(., $regex)"` anschlägt; mit `$regex` als Platzhalter für die jeweiligen regulären Ausdrücke. Dieses Vorgehen wurde bei den Prüfregeln *ZahlFehler*, *UmlautFehler* und *SonstigeKodierungsFehler* (beinhaltet vier `<report>`-Elemente) angewandt.

In Schematron existiert auch ein `<value-of>`-Element, in dem mit dem Attribut `select` ein XPath-Ausdruck übergeben wird, der anhand des Kontextknotens ausgewertet und ausgegeben wird. Dieses Element ist also gleichzusetzen mit dem XSLT-Element `<xsl:value-of>` (s. **Anhang A [6]**).[177] Es gibt keinen zwingenden Grund, das XSLT-Element zu verwenden, allerdings ist dieses im Gegensatz zu Schematrons `<value-of>` z. B. auch in einem `<span>`- oder `<emph>`-Element zulässig.

Die Vorteile der Ausgabe dieser „Fehlerstellen“ bestehen darin, dass das direkte Nachlesen des gemeldeten Fehlers in der Ausgabezeile möglich ist und der Nutzer auf einen Blick sehen kann, ob es sich tatsächlich um einen Fehler handelt. Wirkt es so, als würde es sich um einen Fehler handeln, kann der Nutzer entweder durch einen Blick auf den Textinhalt die Fehlerstelle finden oder alternativ durch eine Textsuche die Stelle finden. Auch für die Entwicklung der Business Rules ist diese Methode hilfreich, da alle

[177] Vgl. *Hedler, M./Montero Pineda, M./Kutscherauer, N.*, Schematron, 2011, S. 139.

gemeldeten Stellen so gefiltert und auf einmal dargestellt werden können. Dies unterstützt die Entwicklung eines Schematron-Schemas dabei, die manuelle Quelldokument-Analyse zu beschleunigen und überschüssige Fehlermeldungen zu vermeiden, die toleriert werden könnten.

6.2 Erläuterung aller Schematron-Regeln

Im Folgenden wird der grundsätzliche Aufbau jedes Patterns in Schematron für ONIX-2.1-Dateien erläutert. Dazu gehören das `<rule>`-Element mit den zu prüfenden Kontextknoten sowie der Knotentest. Bei einigen Prüfregeln bietet sich eine XSL-Transformation an, mit welcher Zeichenketten im ONIX-Quelldokument analysiert werden können, wie es zuvor im **Abschnitt 6.1.4** dargelegt wurde. Zur anschaulichen Erklärung der Erstellung der Patterns werden im Folgenden bei einigen Listings die XSLT-Instruktionen verkürzt dargestellt, worauf jedoch hingewiesen wird. Wie bereits erwähnt, kann in **Anhang A** ein Schematron-Schema mit allen folgenden Patterns in der gleichen Reihenfolge inklusive der vollständigen XSLT-Instruktionen nachgesehen werden.

6.2.1 Erläuterung der Patterns der Kategorie „Falsche Verwendung“

Im Pattern ***EditionsbeschreibungFehler*** (s. **Listing 7**) wird die korrekte Verwendung vom Element `<b058>` überprüft. Dabei soll eine vollständige, freitextliche Beschreibung der Auflage enthalten sein, und nicht bloß eine Ergänzung zu anderen Elementen. Von falschen Elementinhalten, wie etwa „2022“, „vollständig überarbeitete Ausgabe“ (hier fehlt die Nummer der Auflage) oder „Ausgabe mit 2 Bändern“ wurde abgeleitet, was dieses Element zur korrekten Verwendung beinhalten sollte. So beinhaltet das `<assert>`-Element als Bedingung einen regulären Ausdruck als Test, der auf eine Zeichenkette des Elementinhalts passen muss. Mit union-Operatoren werden in einem Regex mehrere mögliche Ausdrücke im Text geprüft:

- der String muss eine Ziffer und darauffolgend einen Punkt haben (z. B. „3. Auflage“)[178] oder
- der String muss eine Ziffer und darauffolgend ein englisches Suffix für eine Zählung enthalten oder
- der String muss eine deutsche wörtliche Zählung enthalten.

Dabei ist im dritten Argument `der matches()`-Funktion mit dem Flag `i` definiert **[1]**, dass die Groß- und Kleinschreibung im untersuchten String ignoriert wird. Da hier freier Text untersucht wird, dieser teilweise sehr unterschiedlich ist und es möglich ist, dass

[178] Auch wenn bei „3. Auflage“ auch nichts Näheres zur Edition beschrieben wird, so soll diese Beschreibung toleriert bleiben, da es zumindest eine freitextliche Beschreibung der Auflage ist.

hier doch kein Fehler vorliegt, ist es für die Benutzerfreundlichkeit vorteilhaft, die Fehlerstelle im Fehlermeldungstext direkt auszugeben **[2]**.

```
<pattern id="EditionsbeschreibungFehler">
    <rule role="error" context="b058">
      <assert
        test="matches(.,
'\d[.]|\d(st|nd|rd|th)|(erst|zweit|dritt|viert|fünft|sechst|
siebt|acht|neunt|zehnt)', 'i')">                                    [1]
        Die Beschreibung in b058 (EditionStatement) ist
möglicherweise unvollständig oder fehlerhaft.
Fundstelle(n): <xsl:value-of select="."/>                           [2]
      </assert>
    </rule>
  </pattern>
```

Listing 7: EditionsbeschreibungsFehler

Bei der Prüfregel ***AbbildungsnotizZiffer*** (s. **Listing 8**) wird im `<report>`-Element mit einer XPath-Konvertierungsfunktion[179] getestet, ob in `<b062>` ausschließlich Zahlzeichen vorkommen, indem geprüft wird, ob der Elementinhalt als integer-Datentyp übersetzbar ist **[1]**.

Beim Pattern ***UntertitelAuflage*** (s. **ebd.**) wird mit der XPath-Funktion `contains()` getestet, ob im Kontextemelement `<b029>`, das für einen Untertitel vorgesehen ist, der Teststring „Auflage“ vorkommt **[2]**. Dieses Wort war in den untersuchten ONIX-Dokumenten immer enthalten, falls die Auflage als Untertitel angegeben wurde, wie z. B. bei „2., aktualisierte Auflage“.

In der Prüfregel ***AbbildungsnotizSeiten*** (s. **ebd.**) wird mit einem Regex-Vergleichsmuster geprüft, ob ein „S“ gefolgt von einem Punkt im Element `<b062>` enthalten ist **[3]**. Seitenzahlangaben wurden in den untersuchten ONIX-Meldungen in diesem Element immer in der Form „S.“ angegeben.

---

[179] Vgl. *Mangano, S.*, XSLT-Kochbuch, 2006.

```
  <pattern id="AbbildungsnotizZiffer">
    <rule role="error" context="b062">
      <report test=". castable as xs:integer">                    [1]
        In b062 (IllustrationsNote) steht nur eine Nummer,
obwohl hier eine Anmerkung stehen soll. Vorschlag: Überführe die
Zahl in b125 (NumberOfIllustrations).
      </report>
    </rule>
  </pattern>

  <pattern id="UntertitelAuflage">
    <rule role="error" context="b029">
      <report test="contains(., 'Auflage')"> In b029 (Subtitle)   [2]
steht die Auflage. Vorschlag: Überführe den Text in b058
(EditionStatement).
      </report>
    </rule>
  </pattern>

  <pattern id="AbbildungsnotizSeiten">
    <rule role="warning" context="b062">
      <report test="matches(., 'S[.]')">                          [3]
        Befindet sich eine Seitenzahlangabe in b062
(IllustrationsNote)?
        Vorschlag: Überführe die Seitenzahlangabe in b061
(NumberOfPages).
      </report>
    </rule>
  </pattern>
```

Listing 8: AbbildungsnotizZiffer, UntertitelAuflage und AbbildungsnotizSeiten

Das Schematron-Pattern ***NonbookmeldungV21*** (s. **Listing 9**) enthält zwei Prüfregeln. Da bei beiden Regeln die gleichen Kontextknoten zu testen sind, kann eine `<rule>` verwendet werden, welcher zwei `<report>`-Elemente mit verschiedenen Tests untergeordnet sind. Hier werden mehrere Kontextknoten untersucht, die die Produktform mithilfe eines zweistelligen Codes angeben, wie z. B. `<b012>ZA</b012>`. Die zweistelligen Codes und ihre Bedeutungen sind in *List 7* (Product Form code) aufgeführt, in der beispielsweise der Code `ZA` für allgemeine Werbeartikel steht, die nicht näher spezifiziert sind. In den ONIX-Dateien wurde untersucht, welche Codes verwendet wurden, um Nonbookprodukte zu kennzeichnen, wobei auffiel, dass für Figuren, Sticker oder Kerzen Codes gewählt wurden, die sehr allgemein „sonstige“ Produkte definieren, obwohl es teilweise genauere Codes gibt, wie etwa *PP* für Sticker. Die genannten Produktform-Codes, die in den untersuchten ONIX-Meldungen bei tatsächlichen Nonbookprodukten enthalten waren, werden im Schematron-Schema als Prädikate zum Kontextknoten angegeben, während die verschiedenen Kontextknotenangaben durch union-Operatoren (|) aneinandergereiht werden **[1]** und als Werte im `context`-Attribut angegeben. Damit Metadatenmanager auch schnell weitere Nonbookprodukte in einer ONIX-Meldung entdecken können, wurden weitere Codes hinzugefügt, die ihre Produkte nicht genauer spezifizieren, wie z. B. Code `00` für

Undefined **[2]**. Im ersten `<report>`-Test bei *NonbookmeldungV21* wird grundsätzlich überprüft, ob dieses Element mit den zuvor angegebenen Codes enthalten ist, damit Metadatenpfleger darauf hingewiesen werden, dass hier kein Buchprodukt vorliegt. Zusätzlich wird mit

```
and not(parent::product/title[matches(., 'spiel|quiz|game', 'i')])
```

ausgeschlossen, dass auf ein Kontextelement zwei Fehlermeldungen zutreffen: Der zweite Test prüft nämlich auch, ob ein Ausdruck des Kontextknotens vorliegt, aber testet zusätzlich, ob im Titel oder Untertitel die Zeichenketten „spiel“, „quiz“ oder „puzzle“ enthalten sind. Mit diesen Schlüsselwörtern im Titel, bei welchen die Groß- und Kleinschreibung ignoriert wird, soll darauf hingewiesen werden, dass es sich explizit um ein Spiel bzw. Quiz oder Puzzle handeln könnte. Außerdem lautet der korrekte Code für Spiele `ZE`.[180]

```
  <pattern id="NonbookmeldungV21">
    <rule context="
        product/b012[. = 'BZ'] | product/b012[. = 'PZ'] |          [1]
product/b012[. = 'ZA'] |
        product/b012[. = 'ZZ'] | product/b012[. = '00']">          [2]
      <report role="information"
        test="exists(.) and not(parent::product/title[matches(.,
'spiel|quiz|game', 'i')])">
        Es handelt sich möglicherweise um ein Nonbookprodukt,
das ungenau kategorisiert wurde.
        Vorschlag: Passe in b012 (ProductForm) den Code an.
      </report>
      <report role="warning"
        test="parent::product/title[matches(.,
'spiel|quiz|game', 'i')]">
        Handelt es sich bei dem Produkt um ein Spiel?
        Vorschlag: In b012 (ProductForm) den Code ZE nutzen.
      </report>
    </rule>
  </pattern>
```

Listing 9: NonbookmeldungV21

Bei dem Pattern ***UntertitelErweiterung*** (s. **Listing 10**) wird geprüft, ob in einem Kontextknoten `<b029>`, der für den Untertitel vorgesehen sind, der Vergleichsstring „Book“ enthalten ist. In vielen ONIX-Meldungen waren hier Werbe- oder Zusatzinformationen festgehalten, wie etwa „plus E-Book inside“. Da immer E-Books als Marketinginformation notiert waren und es sich nicht um einen Untertitel des Buchprodukts handelt, wird nach der Zeichenkette „Book“ gesucht **[1]**, um ggf. auch „eBook“ aufzuspüren.

[180] Vgl. *EDItEUR*, ONIX for Books: Codelists Issue 36, 2017, List 7, Code PP.

```
  <pattern id="UntertitelErweiterung">
    <rule role="warning" context="b029">
      <report test="contains(., 'Book')">                    [1]
        In b029 (Subtitle) stehen möglicherweise
Marketinginformationen.
        Vorschlag: Überführe die Information in ein Other text
composite.
      </report>
    </rule>
  </pattern>
```

Listing 10: UntertitelErweiterung

6.2.2 Erläuterung der Patterns der Kategorie „Biografiefehler“

In der Prüfregel ***ContributorGesamtbio*** (s. **Listing 11**) wird das Element als Kontextknoten angegeben, das die Gesamtbiografie darstellt. Die Gesamtbiografie wird definiert durch das Element `<d102>` mit dem Code `13` für *Biograhical Note* und der Text steht schließlich in `<d104>` **[1]**. In der ONIX-Spezifikation wird angegeben, dass es in diesem Biografietext nicht nur um einen Mitwirkenden gehen soll.[181] Für die Regel wird im `context`-Attribut in Prädikatangaben gezählt, ob mehr als ein `<contributor>` aufgelistet ist und jeder dieser `<contributor>`-Elemente eine `<b044>` als Kind enthält. Innerhalb des `<report>`-Elements wird zunächst getestet, ob im selben Produkt des Kontextelements die `<contributor>`-Container Einzelbiografien enthalten. Daraufhin wird der Elementinhalt von `<b044>` mit dem Elementinhalt des Kontextknotens, also der gefundenen Gesamtbiografie, verglichen. Dafür werden auch die Weißraumzeichen normalisiert. Sind die beiden Elementinhalte gleich, so ist der boolesche Wert des Tests *true* und der Report wird ausgelöst.

Der Fall, dass für ein Buchprodukt ein Contributor angegeben ist, und die Gesamtbiografie sich auf diesen bezieht, könnte toleriert werden, da zumindest nicht andere Contributoren außen vor gelassen werden können. Allerdings könnte es zu Problemen führen, wenn für einen Contributor eine Einzelbiografie sowie eine Gesamtbiografie vorliegt und die Texte sich unterscheiden. Diesen Fall spürt die Prüfregel ***EinContributorBios*** (s. **ebd.**) auf. Der zu prüfende Ausdruck im `context`-Attribut unterscheidet sich im zuvor erläuterten Pattern nur dadurch, dass die Anzahl der `<contributor>`-Elemente in einem `<product>` gleich 1 ist **[2]**. Hier wird nun in Form einer Annahme geprüft, ob sich die Elementinhalte wie in *ContributorGesamtbio* in `<b044>` und `<d104>` gleichen. Gibt dieser Test den Wert *false* aus – also gleichen die Texte sich nicht – so wird eine Fehlermeldung ausgelöst.

---

[181] Vgl. *EDItEUR*, ONIX for Books: Codelists Issue 36, 2017, List 33, Code 13.

```
<pattern id="ContributorGesamtbio">
    <rule role="error" context="
        product[count(contributor) &gt; 1]
        [contributor/b044]
        /othertext[d102 = '13']/d104">                                  [1]
      <report
test="ancestor::product[1]/contributor[b044]/b044/normalize-
space() = normalize-space(.)">
        Der Biografietext eines Contributors stimmt mit dem Text
für alle Contributoren im Other text composite überein,
        obwohl es darin aber um mehrere Contributoren gehen
sollte.
        Vorschlag: Der Biografietext im Other text composite
sollte gelöscht werden.
      </report>
    </rule>
  </pattern>

  <pattern id="EinContributorBios">
    <rule role="error" context="
        product[count(contributor) = 1]                                 [2]
        [contributor/b044]
        /othertext[d102 = '13']/d104">
      <assert
test="ancestor::product[1]/contributor[b044]/b044/normalize-
space() = normalize-space(.)">
        Es gibt nur einen Contributor und dessen Biografietext
weicht vom Text für alle Contributoren ab. Wenn es nur einen
        Contributor gibt, benötigt es keine Gesamtbiografie im
Other text composite.
      </assert>
    </rule>
  </pattern>
```

Listing 11: ContributorGesamtbio und EinContributorBios

Im Pattern ***BioName*** (s. **Listing 12**) wird eine `<let>`-Variable verwendet **[1]**, um den Ausdruck im `test`-Attribut übersichtlich zu gestalten. Durch das `<let>`-Element wird eine Variable mit dem Namen „b036“ erzeugt, die als Wert den Elementinhalt von `<b036>` – also den Namen des Mitwirkenden – mit normalisierten Weißraumzeichen annimmt. Auf den Knoten, die die Einzelbiografie beherbergt **[2]**, wird schließlich die Annahme getestet, ob die erzeugte Variable auf einen darin enthaltenen String passt **[3]**. Innerhalb vom Fehlermeldungstext wird mit `$b036` der Variablenwert, also der Name des Contributors, aufgerufen **[4]**, und als Text wiedergegeben.

```
  <pattern id="BioName">
    <rule role="error" context="b044" >                                    [2]
      <let name="b036"                                                     [1]
        value="ancestor::contributor/b036/normalize-space()"/>
      <assert test="matches(., $b036)">
        Der Name des Contributors muss im Biografietext                    [3]
vorkommen und sollte mit dem angegebenen Namen ('<xsl:value-of
select="$b036"/>') übereinstimmen.                                         [4]
      </assert>
    </rule>
  </pattern>
```

Listing 12: BioName

In der Prüfregel ***BioURL*** (s. **Listing 13**) wird der Kontextknotens `<b044>` darauf getestet, ob er Vergleichsstrings enthält, die charakteristisch für URLs sind, wie „www.“, „.de“ oder „.com“.

Im Pattern ***AutorenBios*** (s. **ebd.**) wird die Anzahl von Autoren und Einzelbiografien in einem `<product>` untersucht, um bei ONIX-Verarbeitungsprozessen den Fall zu vermeiden, dass es mehr als einen Autor gibt und nur die Einzelbiografie eines Autors in die Gesamtbiografie extrahiert wird. Ob ein Contributor ein Autor ist, wird in ONIX mit `<b035>` mit dem Code `A01` für „Author of a textual work“[182] angegeben. Als Kontextknoten wird hier eine Einzelbiografie ausgewählt, die in einem `<product>` enthalten ist, das mehr als einen als Autor definierten Contributor besitzt **[1]**. In `<report>` wird anschließend getestet, ob die Anzahl an Autoren ungleich der Anzahl an Einzelbiografien ist **[2]**. Ist dieser Wert *true*, so wird der Report ausgelöst. Hier bietet es sich an, da bei einem Report mehr als zwei Elemente betrachtet werden müssen, den Contributor-Namen aus `<b036>` als Text auszugeben.

[182] *EDItEUR*, ONIX for Books: Codelists Issue 36, 2017, List 17, Code A01.

```
  <pattern id="BioURL">
    <rule role="error" context="b044">
      <report test="contains(., 'www.') or contains(., '.de')
            or contains(., '.com')">
        Eine URL sollte nicht in einer Biographical note stehen.
        Vorschlag: Nutze dafür das Website composite.
      </report>
    </rule>
  </pattern>

  <pattern id="AutorenBios">
    <rule role="warning" context="
        product[count(./contributor[b035 = 'A01']) &gt; 1]          [1]
        /contributor[b035 = 'A01']/b044">
      <report test="ancestor::product[1]
            [count(./contributor[b035 = 'A01'])
            != count(./contributor[b035 = 'A01']/b044)]">          [2]
        Es gibt ungleich viele Autoren und Biografien in b044
(BiographicalNote). Dies kann zu Fehlern in der Gesamtbiografie
im Other text composite führen.
      </report>
    </rule>
  </pattern>
```

Listing 13: BioURL und AutorenBios

6.2.3 Erläuterung der Patterns der Kategorie „Textliche Fehler“

Wie bereits in **Abschnitt 6.1.4** erwähnt, beinhaltet die Prüfregel ***FehlendesSpace*** Ausnahme-Strings, die zwar auf generische reguläre Ausdrücke passen, aber keine Fehlerreports auslösen sollen. Dieses vollständige Pattern ist in **Anhang A** bei **[7]** zu sehen, während es folgend in **Listing 14** zum einfacheren Verständnis verkürzt dargestellt ist. In *FehlendesSpace* werden in `<rule>` mittels eines union-Operators zwei Kontextknoten angegeben, die typischerweise längere freie Textinhalte besitzen: `<d104>`, das Teil des `<othertext>`-Verbunds ist, und `<b044>`. Im Knotentest des `<report>` wird überprüft, ob das Muster „ein Kleinbuchstabe, darauffolgend ein Punkt, Doppelpunkt, Ausrufezeichen oder Fragezeichen, darauffolgend ein Großbuchstabe“ auf eine Zeichenkette des Kontextknotens passt. Dafür reicht bereits der Regex `'\p{Ll}[.:!?]\p{Lu}'` aus. Bei dieser Prüfregel lohnt sich eine Text-Ausgabe der Fundstelle, da der Fehler öfter auftauchen kann und es sich nicht mit Sicherheit um einen Fehler handelt. Der reguläre Ausdruck wurde jedoch jeweils zu Beginn und zum Ende um „einen, mehrere oder auch keine Wortzeichen“ (`\w*`) ergänzt **[1]**. Dies ermöglicht, dass bei der Referenzierung der Zeichenketten **[2]** längere Zeichenketten inkl. etwas Kontext ausgegeben werden können. Da Weißraumzeichen nicht mehr zu Wortzeichen gehören, wird die Ausgabe der Zeichenkette dadurch begrenzt. Während `'\p{Ll}[.:!?]\p{Lu}'` also nur „e?W“ ausgibt, gibt `'\w*\p{Ll}[.:!?]\p{Lu}\w*'` demnach „Demokratie?Wie“ aus.

```
<pattern id="FehlendesSpace">
    <rule role="warning" context="d104 | b044">
      <let name="Punkt-Regex"
value="'\w*\p{Ll}[.:!?]\p{Lu}\w*'"/>
      <xsl:variable name="VPunkt" as="xs:string*">
        <xsl:analyze-string ...
      </xsl:variable>
      <report test="exists($VPunkt)"> Fehlt ein Leerzeichen
zwischen zwei Sätzen?
        Fundstelle(n): <xsl:value-of select="string-
join($VPunkt, ', ')"/>
      </report>
    </rule>
  </pattern>
```
[1]

[2]

Listing 14: FehlendesSpace

Beim Pattern ***ZahlFehler*** (s. **Listing 15**, verkürzt dargestellt) wird nur in den Elementen `<d104>` nach den Kodierungsfehlern gesucht, bei denen sich ein Fragezeichen zwischen zwei Zahlen befindet, da nur in diesen Elementen dieser Fehler zu finden war. Mit dem Muster des regulären Ausdrucks „mindestens eine Zahl, darauffolgend ein Fragezeichen, darauffolgend mindestens eine Zahl“ wird durch den Quantifizierer `+` sichergestellt **[1]**, dass später bei einem Report die komplette Zahl über ausgegeben wird. Das vollständige Pattern kann in **Anhang A** bei **[8]** nachgesehen werden.

Mit der Prüfregel ***Umlautfehler*** (s. **ebd.**, verkürzt dargestellt) sollen Fehler gefunden werden, die durch eine fehlerhafte Kodierung bei deutschen Umlauten wie *ä*, *ö* oder *ü* aufgetreten sind, wie etwa bei „fu?r“. Es werden zwei mit union-Operatoren verknüpfte Kontextknoten in `<rule>` angegeben, in denen sich dieser Fehler einschleichen könnte: das `<contributor>`-Element, in dem sich Kindelemente mit den Namensangaben der Contributoren sowie die Einzelbiografie befinden und das für Text vorgesehene Element `<d014>` **[2]**. Der in `test` angegebene reguläre Ausdruck überprüft, ob auf einen Vokal als Groß- oder Kleinbuchstabe ein Fragezeichen folgt, auf das direkt ein Kleinbuchstabe folgt. Da es sich nur möglicherweise um einen Umlautfehler handelt, ist auch hier eine Kopie der entsprechenden Textstelle in den Fehlerreport hilfreich. Um das ganze Wort mit dem Fehler auszugeben, wurde der reguläre Ausdruck wieder durch das Muster „einen, mehrere oder auch keine Wortzeichen“ eingegrenzt **[3]**. Das vollständige Pattern ist in **Anhang A** bei **[9]** zu finden.

```
  <pattern id="ZahlFehler">
    <rule role="warning" context="d104">
      <let name="ZahlFehler-Regex" value="'\d+[?]\d+'"/>                [1]
      <report test="matches(., $ZahlFehler-Regex)">
        <xsl:variable name="VZahlFehler" as="xs:string*">
          <xsl:analyze-string ...
        </xsl:variable> In einer Zahl liegt möglicherweise ein
Fehler vor.
        Fundstelle(n): <xsl:value-of select="string-
join($VZahlFehler, ', ')"/>
      </report>
    </rule>
  </pattern>

  <pattern id="UmlautFehler">
    <rule role="warning" context="contributor | d104">                 [2]
      <let name="UmlautFehler-Regex" value="'\w*[aouAOU][?][a-         [3]
z]\w*'"/>
      <report test="matches(., $UmlautFehler-Regex)">
        <xsl:variable name="VUmlautFehler" as="xs:string*">
          <xsl:analyze-string ...
        </xsl:variable> Möglicherweise liegt ein Umlautfehler
vor.
        Fundstelle(n): <xsl:value-of select="string-
join($VUmlautFehler, ', ')"/>
      </report>
    </rule>
  </pattern>
```

Listing 15: FehlendesSpace, ZahlFehler und UmlautFehler

Im Pattern ***SonstigeKodierungsfehler*** (s. **Listing 16**, verkürzt dargestellt) befinden sich vier `<report>`-Elemente, die alle einem gemeinsamen `<rule>`-Element untergeordnet sind und sich somit auf die gleichen Kontextknoten `<d104>` oder `<b044>` beziehen, die üblicherweise viel freien Text beinhalten. In diesem Listing sind die `<report>`-Elemente zur anschaulicheren Erklärung verkürzt abgebildet; in **Anhang A** bei **[10]** kann das vollständige Pattern nachgesehen werden. In den `<report>`-Tests werden unterschiedliche Vergleichsmuster auf die Zeichenketten der Kontextknoten überprüft; dabei werden die verschiedenen Tests nur zur Übersichtlichkeit in unterschiedliche `<report>`-Elemente gepackt. Diese Reports wurden auch erstellt, um nicht andere deutlicher zu kategorisierenden Kodierungsfehler doppelt zu melden, wie etwa *ZahlFehler* oder *FehlendesSpace*. In den Kommentaren in **Listing ebd.** sind Beispiele für die Kodierungsfehler zu finden.

Der erste Test „A“ **[1]** prüft, ob ein String vorhanden ist, in dem nach einem Whitespace ein Fragezeichen kommt. So werden alle Strings gefunden, in denen z. B. ein Leerzeichen vor einem Fragezeichen erscheint. Für eine Ausgabe des Textes im Fehlerreport wird zu Beginn des Vergleichsstrings ergänzt, dass Wortzeichen beliebig oft voranstehen können. Damit die Fehlerstellen mehr Text beinhalten, um per Suchbefehl in der ONIX-Datei gefunden werden zu können, wird der Regex nach hinten um ein optionales, beliebig oft auftretendes Nicht-Wortzeichen ergänzt, auf das beliebig oft weitere Wortzeichen folgen können.

Der Test „B“ **[2]** prüft, ob direkt auf ein Fragezeichen mindestens ein kleiner Buchstabe folgt. Um sich nicht mit den Fehlermeldungen von *UmlautFehler* zu überschneiden, werden vor dem Fragezeichen die Vokale negiert, ansonsten sind alle Zeichen möglich. Mit dem Flag `s` in der Funktion erlaubt der Punkt im Regex ebenso das Zeilenumbruchzeichen vor dem Fragezeichen. Für eine Ausgabe von etwas Kontext sind beliebig viele Wortzeichen zu Beginn erlaubt.

Im `<report>`-Element „C“ **[3]** wird der Kontextknoten auf das Zeichenmuster geprüft, dass auf einen Punkt ein Fragezeichen folgt. Da diese Prüfregel in „B“ bereits greift, muss hier mindestens ein Großbuchstabe auf das Fragezeichen folgen. Für eine Textausgabe mit etwas Kontext sind wieder beliebig viele Wortzeichen zu Beginn des Regex erlaubt.

Im vierten Test „D“ **[4]** wird getestet, ob im Kontextknoten auf ein Nicht-Zahlzeichen ein Fragezeichen folgt, auf das mindestens ein Zahlzeichen folgt. Durch die Negierung `\D` wird sichergestellt, dass diese Prüfregel die *ZahlFehler*-Prüfregel nicht überschneidet. Für eine Ausgabe mit Kontext wird zu Beginn des Regex noch genau ein beliebiges Zeichen erlaubt.

```
  <pattern id="SonstigeKodierungsfehler">
    <rule role="warning" context="d104 | b044">

      <!-- A: Bsp.: 2018 ? 1 oder der ?Hirnstamm-->
      <report test="matches(., '\w*\s[?]\W\w*')">                    [1]
        A: Hier liegen möglicherweise Zeichenfehler mit einem
Whitespace vor.
      </report>
      <!-- B: Bsp.: 113?ff -->
      <report test="matches(., '\w*.[^a^o^u^A^O^U][?]\p{Ll}+',       [2]
's')">
        B: Hier liegen möglicherweise Kodierungsfehler vor.
      </report>
      <!-- C: Bsp.: St.?Gallen -->
      <report test="matches(., '\w*\.[?]\p{Lu}+', 's')">             [3]
        C: Hier liegen möglicherweise Kodierungsfehler vor.
      </report>
      <!-- D: Bsp.: §?175 oder kann?10 oder S.?75 -->
      <report test="matches(., '.?\D[?]\d+', 's')">                  [4]
        D: Hier liegen möglicherweise Kodierungsfehler vor.
      </report>
    </rule>
```

Listing 16: SonstigeKodierungsfehler

Während des Erstellens der Prüfregeln wurde sich dafür entschieden, die beiden Prüfregeln ***Bindestrich*** und ***DreiBuchstaben*** aus Effizienzgründen nicht in das Schematron-Schema aufzunehmen. Beide Patterns zeigten nach einer Validierung teilweise sehr viele Fehlerstellen an, von welchen nur ein kleiner Teil tatsächliche Fehler darstellte. So meldete das Pattern *Bindestrich* 45 Items, in denen jeweils bis zu 20 Zeichenketten ausgegeben wurden, von welchen jedoch nur fünf Zeichenketten eindeutige Bindestrichfehler enthielten. Das Pattern *DreiBuchstaben* meldete fünf

Items, von welchen mit zwei echten Fehlern weniger als die Hälfte tatsächliche Fehler darstellte, die optisch kaum auffielen (z. B. „solllte“ statt korrekt „sollte“). Das Weglassen der Patterns steigert zum einen die Performance, da der Prozessor weniger ablaufen muss, und sorgt zum anderen für eine übersichtlicheren Gesamtfehlerreport.

6.2.4 Erläuterung der Patterns der Kategorie „Empfehlungen“

Bei der Prüfregel ***SeitJahren*** (s. **Listing 17**) wird in den für Biografien verwendeten Elementen, die mit einem union-Operator in `context` angegeben sind **[1]**, geprüft, ob ein Vergleichsstring nach dem Muster „seit … Jahren“ vorhanden ist. Im regulären Ausdruck ist dabei mit dem Flag `i` **[2]** definiert, dass die Groß- und Kleinschreibung ignoriert wird und dass die Zählung auch in wörtlicher Form stattfinden kann. Passt der Regex auf eine Zeichenkette im Kontextknoten, so wird der Report ausgelöst, in dem auch der Vorschlag notiert ist, den Textabschnitt in die Formulierung „seit dem Jahr …“ oder „seit über … Jahren“ umzuändern.

Im Pattern ***KeywordInflation*** (s. **ebd.**) wird in einem `<product>` die Anzahl der Keywords untersucht. Als Kontextknoten wird das erste `<subject>`-Element in einem `<product>` gewählt **[3]**, da ein `<subject>`-Element jeweils ein Keyword als Kind enthält. Dadurch, dass bei einem ausgelösten Report das erste `<subject>`-Element als Fehlerreferenz angegeben bzw. im Oxygen XML Editor unterschlängelt wird, können die folgenden Keywords auf diese Weise gut betrachtet werden. Es wird getestet, ob im selben Produkt mehr als 35 Keywords gelistet sind, welche innerhalb von `<subject>` im Element `<b067>` mit dem Code `20` für Keywords definiert sind **[4]**.

```
<pattern id="SeitJahren">
    <rule role="warning"
      context="b044 | othertext[d102 = '13']/d104">                [1]
      <report
        test="matches(.,
'seit[\p{Zs}\s]+(\d+|zwei|drei|vier|fünf|sechs|sieben|acht|
neun|zehn|elf|zwölf)[\p{Zs}\s]+Jahren', 'i')">                    [2]
        Die Information in der Biografie kann veraltet sein.
        Vorschlag: „seit dem Jahr ..." oder „seit über ...
Jahren".
      </report>
    </rule>
  </pattern>

  <pattern id="KeywordInflation">
    <rule role="information" context="product/subject[1]">         [3]
      <report test="ancestor::product[count(./b067/'20') &gt;      [4]
35]"> Reduziere die Keywords. Es sind über 35 Keywords zu
        diesem Produkt eingetragen.
      </report>
    </rule>
  </pattern>
```

Listing 17: SeitJahren und KeywordInflation

Mit der Prüfregel ***Zeichenlaenge*** (s. **Listing 18**) wird im Knoten `<b203>` die Zeichenlänge des Elementinhalts darauf getestet, ob sie 300 Zeichen überschreitet. Diese Schematron-Prüfregel eignet sich besonders gut dafür, sie individuellen Bedürfnissen anzupassen und Wunschlängen bestimmter Elemente zu prüfen.

Im Pattern ***TOCDefaultTextFormat*** (s. **ebd.**) werden die `<d104>`-Elemente untersucht, die als Inhaltsverzeichnis definiert sind **[1]**, gleichzeitig als Textformat *default* angegeben haben **[2]** und somit nicht genauer spezifiziert sind. Dass es sich um ein TOC handelt, wird in `<d102>` mit dem Code `04` für *Table of contents* angegeben. Es besteht alternativ die Möglichkeit, das Textformat in `<d104>` mit dem ONIX-Attribut `textformat` anzugeben; dabei entspricht der Attributwert dem gleichen Code, wie in *List 33* **[3]**. In der Dokumentation der Codelist wird auch nicht ausgeschlossen, dass ein Default-Textformat verwendet werden darf, allerdings werden Alternativvorschläge unterbreitet.[183] Wurde in den untersuchten ONIX-Meldungen das Textformat mit dem Code `06` als *default* angegeben, so kam es häufig vor, dass der entsprechende Text unstrukturiert auf Buchhandel-Websites zu finden war. Laut ONIX-Spezifikation sind Spitzklammern in diesem Textformat nicht erlaubt, Entitäten wie `&amp;` und `&lt;` hingegen schon. In diesem Zusammenhang war gelegentlich in den ONIX-Meldungen `&lt;br/&gt;` notiert, was in *escapter* Schreibweise für das leere Element `<br />` steht und in XHTML einen Zeilenumbruch erzeugt.[184] Wurden diese Texte im WWW abgebildet, so waren diese immerhin mit Zeilenumbrüchen im Text und somit geordneter dargestellt. Aus diesem Grund wird im `<report>`-Test eine Ausnahme für das Muster „br /“ bzw. „br/“ formuliert **[4]**. Somit wird ein Report ausgelöst, falls ein TOC im Default-Textformat vorliegt, jedoch wird kein Report ausgelöst, falls sich im Kontextknoten mindestens ein Umbruchszeichen befindet. Wie bereits in **Abschnitt 5.3** erwähnt, können zusätzliche Prüfungen auf Grundlage weiterer HTML- bzw. XHTML-Elemente[185] oder Codes erstellt werden, die vom Aufbau der Prüfregel *TOCDefaultTextFormat* abgeleitet werden können. Es kann zudem überlegt werden, die Prüfung auf Vorhandensein von escaptem HTML wieder aus der Regel herauszunehmen, da dies indiziert, dass im Grunde kein reiner Text vorliegt.

---

[183] Vgl.*EDItEUR*, ONIX for Books: Codelists Issue 36, 2017, List 33, Code 04.
[184] Vgl. *HTML-Seminar.de*, Zeilenumbruch erzwingen über <br>, 2023.
[185] Vgl. *EDItEUR*, ONIX for Books: Product Information Message, 2021, S. 2 f.

```
  <pattern id="Zeichenlaenge"><!-- individuelle Empfehlungen oder
Wünsche auch mgl. -->
    <rule role="information" context="b203">
      <report test="./string-length() &gt; 300">
        Die empfohlene Länge beträgt hier 300 Zeichen. Wenn
möglich, kürze den Text.
      </report>
    </rule>
  </pattern>

  <pattern id="TOCDefaultTextFormat">
    <rule role="information" context="othertext[d102 = '04']      [1]
                                      [d103 = '06']/d104      |   [2]
                        othertext[d102='04'][@textformat='06']">  [3]
      <report test="not(matches(., 'br\s*/'))">                   [4]
        Das TOC könnte auf eine Website übernommen werden und so
seine Struktur verlieren.
        Vorschlag: Nutze bei d103 den Code 05 und überführe das
TOC in eine XHTML-Struktur.
      </report>
    </rule>
  </pattern>
```

Listing 18: Zeichenlaenge und TOCDefaultTextFormat

6.3 Erzeugung eines HTML-Reports

Bei der Validierung gegen ein Schematron-Schema werden im Hintergrund die Fehlermeldungen als Output generiert und standardmäßig in ein SVRL-Dokument übernommen. SVRL ist eine medienneutrale XML-Schnittstellensprache, wodurch dieses Protokoll bei einer weiteren Transformation in eine beliebige medienspezifische XML-Syntax überführt werden kann.[186] Dies macht sich beispielsweise auch der Oxygen-XML-Editor zunutze, welcher die im SVRL hinterlegten Informationen wie XPath-Lokalisierung des Fehlers, Schweregrad und Fehlertext nutzt, um die Fehler an Ort und Stelle kenntlich zu machen. Um zu gewährleisten, dass Nutzer des Schematron-Schemas, die eine andere Software nutzen, auch ein nutzerfreundliches Protokoll erhalten, kann mittels einer XSLT-Transformation ein HTML-Report erzeugt werden. Dafür kann sich an einem bereits beim Praxispartner entwickelten XSLT-Skript für einen Schematron-HTML-Report orientiert werden. Einige XSLT-Skripte dazu sind in einem Github Repositorium[187] des Praxispartners le-tex zu finden. So können die Fehlermeldungen in HTML in einer Tabelle ausgegeben werden, in welcher bei jedem ausgelösten Fehlerreport folgendes angegeben wird:

[186] Vgl. *Hedler, M./Montero Pineda, M./Kutscherauer, N.*, Schematron, 2011, S. 117 f.
[187] https://github.com/transpect/schematron/tree/master/xsl

- der Schweregrad
- der Ort des Kontextknotens als XPath-Ausdruck und der entsprechende ONIX-Reference-Tag-Name zum im XPath enthaltenen Short-Tag-Name
- die Fehlermeldung
- die ID des `<pattern>`-Elements.

Da es eine Hürde für die XSL-Transformation darstellt, dass die ONIX-Elemente in Short- oder Reference-Tags vorliegen können, ist es ein Lösungsweg, vor der Transformation zum HTML-Report dafür zu sorgen, dass das ONIX-Dokument in Short-Tag-Syntax überführt wird, falls es nicht schon so vorliegt. Dafür kann eine XSL-Transformation mit Unterstützung des „tagname-converter[s]“[188] generiert werden, der eine ONIX-DTD zur Grundlage nimmt. In dieser DTD sind beide Elementnamensangaben in `refname`-Attributen definiert, was es ermöglicht, im Schematron-Schema diese Attribute zu referenzieren und nach der Validierung im HTML-Report den entsprechenden Reference-Namen auszugeben. Dies wird über eine XSLT-Instruktion über ein `<properties>`-Element (s. **Listing 19**) und eine Referenzierung in den `<report>`- und `<assert>`-Elementen ermöglicht, wie es in **Anhang B** bei **[3]** zu sehen ist. Für ONIX 3.0 existiert ein neues XSLT-Skript für den Tagname-Konverter, das auch online verfügbar ist.[189]

```
<properties>
   <property id="refname">
        <xsl:value-of select="@refname"/>
   </property>
</properties>
```

Listing 19: <properties>-Element für Referenzierung des Reference-Tags

Zusätzlich zu den beiden genannten XSL-Transformation ist vorab eine weitere XSL-Transformation denkbar, um die Voraussetzungen für eine erfolgreiche Validierung zu erfüllen. Bei einem untersuchten Datensatz befanden sich beispielsweise zwei Elemente (`<NumberOfPages>` und `<ProductFormDescription>`) in der falschen Reihenfolge. Damit trotzdem nicht schon bei der DTD-Validierung abgebrochen wird, wird das Dokument per XSLT zunächst so transformiert, dass es DTD-valide ist. Diese Vorprozessierung kann erweitert werden, sollten von anderen Marktteilnehmen regelmäßig ähnlich Unregelmäßigkeiten geliefert werden. So kann ein erstes XSLT-Stylesheet die ONIX-Datei DTD-valide machen, falls eine DTD-Deklaration nicht oder inkorrekt angegeben ist.

Um alle diese Schritte unabhängig von der Arbeitsumgebung einfacher anwendbar zu machen, empfiehlt es sich, eine XSLT-Pipeline zu erstellen, in der diese XSLT-Skripte

[188] *EDItEUR*, ONIX tagname converter, 2023.
[189] https://www.editeur.org/files/ONIX%203/ONIX%203.0%20tagname%20converter%20v1.3.html

nacheinander ablaufen. Kurz zusammengefasst baut sich die XSLT-Pipeline aus Szenarios mit folgenden Zielen zusammen:

1. Die ONIX-Datei wird DTD-valide gemacht.
2. Falls die ONIX-Datei in Reference-Tag-Syntax vorliegt, wird diese in Short-Tag-Syntax umgewandelt.
3. Die ONIX-Datei wird gegen das Schematron-Schema validiert und ein HTML-Report wird erzeugt und angezeigt.[190]

In dieser XSLT-Pipeline (s. **Anhang D**) ist der in **Listing 20** aufgezeigte zweistufige Entscheidungsbaum implementiert, der die DTD bestimmt, die zu verwenden ist. Die Übersetzung dieser URLs zu den lokal hinterlegten DTD-Dateien erfolgt durch den XML Catalog[191].

```
<xsl:variable name="doctype" as="xs:string"
    select="if (/*/@release = '3.0')
            then if (/ONIXMessage)
                 then
'https://www.editeur.org/onix/3.0/reference/ONIX_BookProduct_3.0_refe
rence.dtd'
                 else
'https://www.editeur.org/onix/3.0/short/ONIX_BookProduct_3.0_short.dt
d'
            else if (/ONIXMessage)
                 then
'http://www.editeur.org/onix/2.1/reference/onix-international.dtd'
                 else 'http://www.editeur.org/onix/2.1/short/onix-
international.dtd'"/>
```

Listing 20: Implemetierter Entscheidungsbaum zur Wahl der DTD

[190] https://github.com/transpect/schematron/blob/master/xsl/validate-and-render-svrl.xsl
[191] https://github.com/transpect/schema-onix/blob/master/xmlcatalog/catalog.xml

7 Anwendung der Schematron-Regeln in ONIX 3.0

Wie bereits erwähnt, waren die bedeutendsten Gründe für eine Veröffentlichung von ONIX 3.0 die Abdeckung digitaler Produkte als wichtiger Teil mit Raum für neue Produktformate und die Eliminierung veralteter Elemente, die davor immer beibehalten wurden, um die Abwärtskompatibilität zu gewährleisten. Außerdem gibt es in ONIX 3.0 eine erweiterte Titelzusammensetzung, die einen genaueren Ausdruck von Titeldetails ermöglicht und es existieren mehr Möglichkeiten, verschiedenes Marketingmaterial zu spezifizieren. Insgesamt ist auch lobenswert, dass es einen globalen Implementierungs- und Best-Practice-Leitfaden gibt, statt wie vorher im Wesentlichen nationale Leitfaden, um in möglichst hohem Maße Interoperabilität zu gewährleisten.[192] Dennoch wird auch bei ONIX-3.0-Meldungen von zweckentfremdeten ONIX-Elementen berichtet[193] und prinzipiell sind hier auch textliche Fehler möglich. Im Folgenden wird erläutert, wie ein gemeinsames Schematron-Dokument für ONIX 2.1 und 3.0 erstellt werden kann, und anschließend wird untersucht, ob in ONIX-3.0-Dateien die erstellten Prüfregeln noch in gleichem Maß gelten.

7.1 Erstellung der Schematron-Regeln für ONIX 3.0

Im **Anhang B** ist das vollständige Schematron-Schema für beide ONIX-Versionen abgebildet.

Um überprüfen zu können, ob die Schematron-Prüfregeln auch in ONIX 3.0 erfolgreich anschlagen, muss das Schematron-Schema angepasst werden, da ONIX 3.0, wie bereits erwähnt, technische sowie strukturelle Unterschiede zu ONIX 2.1 beinhaltet. Es ist möglich, mit einigen Mitteln das Schematron-Schema für ONIX 2.1 so anzupassen, dass *eine* Schemadatei auf ONIX-Dateien beider Versionen angewandt werden kann. Dafür muss beachtet werden, dass die Elemente in ONIX 3.0 neuerdings in einem Namespace sind und somit bei den XPath-Ausdrücken auch ein Namespace hinzuzufügen ist. Dies kann mit einer Wildcard für den Namensraum ermöglicht werden, indem vor jedes in Schematron referenzierte ONIX-Element ein `*:` hinzugefügt wird, wodurch sowohl in ONIX 2.1 als auch in ONIX 3.0 Elemente angesprochen werden.

Für die erstellten Business-Rules sind v. a. die in **Tabelle 13** Unterschiede in der XML-Struktur zu beachten, dabei ist links die Verschachtelung in ONIX 2.1 und rechts in ONIX 3.0 zu sehen.

---

[192] Vgl. *EDItEUR*, Frequently Asked Questions about ONIX for Books, 2009.
[193] Vgl. *Herlinger, B.*, Zweckentfremdungen in ONIX 2.1 und 3.0, 2022.

ONIX 2.1	ONIX 3.0
product/contributor	product/descriptivedetail/contributor
product/othertext/d104	product/collateraldetail/textcontent/d104
product/subject/b070	product/descriptivedetail/subject/b070
product/title	product/descriptivedetail/titledetail /titleelement

Tabelle 13: Unterschiede in der Elementstruktur in ONIX 2.1 und 3.0

Innerhalb von <rule context="...">-Elementen kann bei den Fällen, in denen in ONIX 3.0 ein strukturiertes Element wie z. B. <descriptivedetail> hinzugefügt wurde, der XPath-Ausdruck für 2.1 und 3.0 gemeinsam angegeben werden, indem ein doppelter Slash (//) gewählt wird: der Ausdruck, um <contributor>-Elemente zu adressieren, lautet folglich: *:product//*:contributor. Sollte sich der Name eines Kontextelementes in <rule> unterschieden, wie es etwa bei <othertext> bzw. <textcontent> der Fall ist, so können als Wert des Kontextattributes beide Pfade angegeben werden, indem sie mit einem union-Operator getrennt werden: *:othertext | *:textcontent.

Falls sich die XPath-Ausdrücke in <report test="..."> bzw. <assert test="..."> in beiden Versionen sehr stark unterscheiden, ist es eine Option, für beide Versionen jeweils ein Pattern zu erstellen, in welchen die Pfadangaben genau, ohne //, angegeben werden. So wird auch verhindert, dass beide Pattern auf einen Fehler anschlagen. Im Schematron-Schema wurde diese Möglichkeit einmal bei der Business Rule *Nonbookmeldung* angewandt. In **Anhang B** sind bei **[4]** und **[5]** die vollständigen Patterns dazu zu finden. Um im Schematron-Schema die Fehlermeldungen für die unterschiedlichen Versionen besser auseinander halten zu können, wurde sowohl in den ID-Namen als auch in den Fehlermeldungstexten jeweils ein „V2.1“ bzw. „V3.0“ hinzugefügt für *ONIX Version 2.1* bzw. *3.0.* Da die Struktur und Codes bei diesen ONIX-Elementen zwischen den beiden Versionen unterschiedlich sind, unterscheiden sich auch die Lösungsvorschläge, was in **[6]** und **[7]** (s. ebd.) ist zu erkennen ist.

Nützlich ist, dass in der Short-Tag-Syntax alle (vierstelligen) neuen ONIX-3.0-Elemente an der Stelle des Buchstabens den Buchstaben *x* als Kennzeichnung dafür haben, dass es sich hierbei um ein neues Element handelt, wie z. B. <x426> für <TextType>.

7.2 Optimierungen in ONIX 3.0

In diesem Abschnitt wird untersucht, welche der Fehlerquellen und möglichen Zweckentfremdungen von ONIX-Elementen, die zuvor für ONIX 2.1 dargelegt wurden, auch in ONIX 3.0 auftreten können. Insbesondere wird untersucht, ob und wo Verbesserungen vorgenommen wurden und an welchen Stellen die Schematron-Prüfregeln auch für ONIX 3.0 hilfreich sind. Für die Analyse wurden zunächst

vorliegenden ONIX-3.0-Meldungen gegen das Schematron-Schema validiert und beobachtet, welche Fehlermeldungen auftraten. Anschließend wurden bei jeder Regel die getesteten ONIX-Elemente in den ONIX-Spezifikationen beider Versionen gesucht und deren Beschreibung verglichen. Falls Codes aus den Codelisten vorkamen, so wurden auch die Beschreibungen dieser verglichen. Zuletzt wurde in Dokumenten von EDItEUR, in welchen die Änderungen zu ONIX 3.0 beschrieben sind, zu den untersuchten ONIX-Elementen recherchiert sowie mittels einer Textsuche in den HTML-Codelisten beider Versionen durch Schlüsselwörter (z. B. „Table of contents“) nach passenden Codes gesucht und diese verglichen. Zur Analyse werden dieselben Spezifikationen und Codeslists wie in **Abschnitt 4.2.2** erwähnt verwendet. Die Prüfregeln bzw. ONIX-Elemente mit positiven Veränderungen werden im Folgenden zuerst behandelt und absteigend folgen die Elemente, die wenige bis gar keine Optimierung erfahren haben.

Die größte Verbesserung bei den aufgetretenen Problematiken ist beim Metadatenaustausch über Nonbook- bzw. Spielprodukte vorgenommen worden. Hier wurde bei ONIX 3.0 für den `<SubjectSchemeIdentifier>` eine Reihe an neuen Codes in die entsprechende *List 27* aufgenommen. Für den deutschen Markt ist der Code `C7` für „Klassifikationen von Spielen, Puzzles und Spielwaren“[194] hinzugefügt worden, welcher nochmals eine Reihe an Spielekategorien aufführen lässt, die in einem externen PDF-Dokument[195] festgehalten sind. Dennoch ist die in der Prüfregel abgebildete Formulierung mit der Angabe durch unpräzise Codes weiterhin gültig, weshalb die Prüfregel *NonbookmeldungV30* auch weiterhin in ONIX 3.0 Anwendung finden kann.

Auch für die Angabe eines Inhaltsverzeichnisses wurden Änderungen vorgenommen. In **Tabelle 14** sind die Codelists beider ONIX-Versionen untereinander dargestellt und zeigen die Angabe der Codes für TOCs. So wird in ONIX 3.0 ein Table of Contents mit dem neuen Element `<x426>` und dem Code `04` verwendet, wobei eine Angabe des Textformates mit `<d103>` nicht mehr möglich ist, da dieses Element in ONIX 3.0 nicht mehr existiert. Das Textformat kann ausschließlich in Form des ONIX-Attributes `textformat` angegeben werden. Der entsprechende Code aus List 34 gibt dabei das Textformat an, wie etwa `textformat="06"`, das für *Default text format* steht.[196] Im finalen Schematron (s. **Anhang B**) wurde die Prüfregel um diesen Aspekt ergänzt. Das Inhaltsverzeichnis darf weiterhin unstrukturiert dargestellt werden, darf aber auch XHTML-Elemente verwenden, wie es auch in der **Tabelle ebd.** in der untersten Zeile zu sehen ist.

---

[194] *EDItEUR*, ONIX for Books: Codeslists Issue 59, 2022 List 27, Code C7.
[195] https://www.ludologie.de/fileadmin/user_upload/PDFs/211126_Kategorisierung_von_Spielen_Puzzles_und_Spielwaren.pdf (führt zu Download)
[196] https://ns.editeur.org/onix/en/34

<table>
<tr><th></th><th>Auszug für „Table of contents”</th></tr>
<tr><td>Codelists Issue 36 für ONIX 2.1[197]</td><td>
<table>
<tr><th>List</th><th>Code</th><th>Heading</th></tr>
<tr><td>33</td><td></td><td>Other text type code…
Used with <TextTypeCode> <d102></td></tr>
<tr><td>33</td><td>01</td><td>Main description</td></tr>
<tr><td>33</td><td>02</td><td>Short description/annotation</td></tr>
<tr><td>33</td><td>03</td><td>Long description</td></tr>
<tr><td>33</td><td>04</td><td>Table of contents
Used for a table of contents sent as a single text field, which may or may not carry structure expressed through HTML etc. Alternatively, a fully structured table of contents may be sent by using the <ContentItem> composite</td></tr>
</table>
</td></tr>
<tr><td>Codelists Issue 59 für ONIX 3.0[198]</td><td>
<table>
<tr><th>List</th><th>Code</th><th>Heading</th></tr>
<tr><td>153</td><td></td><td>Text type…
Used with <TextType> <x426></td></tr>
<tr><td>153</td><td>01</td><td>Sender-defined text</td></tr>
<tr><td>153</td><td>02</td><td>Short description/annotation</td></tr>
<tr><td>153</td><td>03</td><td>Description</td></tr>
<tr><td>153</td><td>04</td><td>Table of contents
Used for a table of contents sent as a single text field, which may or may not carry structure expressed using XHTML</td></tr>
</table>
</td></tr>
</table>

Tabelle 14: Vergleich beider Codelists mit Veränderungen für „Table of contents“

In diesem Zusammenhang ist eine weitere positive Veränderung zu erwähnen: das Element `<x427>`, das das Publikum definiert, für den der Textinhalt im `<d104>`-Element bestimmt ist. Dieses Element ist verpflichtend für alle Erscheinungen im `<textcontent>`-Verbund und hilft somit der Einordnung, welche Texte mit großer Wahrscheinlichkeit bei Endverbrauchern zu lesen sein werden und dementsprechend sauber dargestellt werden sollen.

Für die Angabe von Zusatzinformationen für Werbezwecke, wie es bei der Prüfregel *UntertitelErweiterung* im Untertitel angegeben wurde, gibt es in ONIX 3.0 einige Lösungsmöglichkeiten. So kann im Element `<x426>` mit dem Code `10` für *Promotional headline* ein kurzer Text zu Werbezwecken angegeben werden: „A promotional phrase which is intended to headline a description of the product“[199]. Eine weitere Möglichkeit wäre es in diesem Fall, die enthaltenen Zusätze als „Feature“ anzugeben. Dies wird mit dem Code `11` für *Feature* angegeben und ist genau dafür gedacht, Aufmerksamkeit auf das Produkt zu ziehen.

Bezüglich der Prüfregel *SeitJahren* wurde in der ONIX-Spezifikation bei `<b044>` ein Hinweis hinzugefügt, dass zeitsensitive Ausdrücke vermieden werden sollen und dass im Zweifel ein `datestamp`-Attribut hinzugefügt werden kann:

---

[197] https://ns.editeur.org/onix36/en
[198] https://ns.editeur.org/onix/en
[199] *EDItEUR*, ONIX for Books: Codeslists Issue 59, 2022, List 153, Code 10.

> „Beware of biographical notes including phrases such as 'her latest work is…', as they are somewhat time-sensitive, and consider the use of the datestamp attribute if such phrases cannot be avoided"[200].

Für alle Prüfregeln der Klasse „TextFehler" ist die bereits erwähnte verpflichtende Angabe der angedachten Leserschaft des Metadatentextes in <d104> eine kleine Hilfe, da so gezielt überprüft werden kann, welche Textinhalte möglicherweise veröffentlicht werden und dementsprechend korrekt sein müssen. Allerdings gilt die verpflichtende Angabe mit <x427> nicht für <b044> und zu Kodierungsfehlern kann es weiterhin kommen. Eine weitere Unterstützung bietet die Spezifikation, indem sie bei Elementen, die die Verwendung von XHTML, HTML oder XML gestatten, einen sichtbaren Link mit Informationen dazu in der Spezifikation bieten.[201] In ONIX 3.0 ist auch beim Element <b062> sowie <b058> XHTML aktiviert. Außerdem wurde bei der Beschreibung der Verwendung von <b058> deutlicher angegeben, dass korrekterweise „edition type and number"[202] für die Indexierung und das Auffinden angegeben sein müssen.

Bezüglich der Prüfregel *BioName* gibt es Ergänzungen in den *Codelists 17* (Contributor role code), *18* (Person / organization name type) und *19* (Unnamed person(s)), welche nun vermeiden können, dass sich der angegebene Name des Autors bzw. der Einrichtung mit der Namensangabe in der Biografie unterscheidet. So ist es etwa auch mit dem neuen Element-Verbund <alternativename> möglich, innerhalb eines <contributor>-Verbundes einen Künstlernamen zum echten Namen anzugeben.[203]

Für Keywordangaben gibt keine Änderungen in der Spezifikation, somit behält die Empfehlungsprüfregel *KeywordInflation* weiterhin ihre Berechtigung. Für das Element <b203> mit der empfohlenen Zeichenlänge hat sich auch nichts geändert, allerdings gab es bei anderen Elementen durchaus Unterschiede bei der empfohlenen Zeichenlänge in ONIX 2.1 und 3.0, wie etwa bei <b336>. Für die Prüfregel *BioURL* hat sich nichts verändert; hier soll weiterhin laut Spezifikation eine URL im Text vermieden werden, um potenzielle Verarbeitungsprobleme zu vermeiden. Die Prüfregeln *ContributorGesamtbio*, *EinContributorBios* und *AutorenBios* behalten auch bei ONIX 3.0 ihre Berechtigung.

---

[200] *EDItEUR*, ONIX for Books: Product Information Format Specification, 2021, S. 93.
[201] Vgl. *ebd.*, S. 28.
[202] *Ebd.*, S. 109.
[203] Vgl. *ebd.*, S. 89 f.

Fazit

Neben zweckentfremdeter und uneinheitlicher Nutzung von ONIX-Elementen können auch weitere Problematiken und Fehler in ONIX-Dateien mit ihrer Identifizierung und anschließender Korrektur die Qualität von den Daten verbessern. Mit der Zunahme des Onlinebuchhandels wächst auch die Wichtigkeit, ausreichende, korrekte und bestenfalls umfangreiche Daten den Endkunden online zur Verfügung zu stellen, um gefunden zu werden und für ein positives Kauferlebnis zu sorgen. Diese Online-Informationen stammen u. a. auch aus ONIX-Metadaten, wie die Recherche zeigte. Wenn in den Metadaten eine uneinheitliche oder inkorrekte Verwendung bestimmter ONIX-Elemente stattfand, so zeigte sich dies oft auf Händler- oder Verlagswebsites. Schematron-Prüfregeln können hier eingesetzt werden, um vorab solche fehlerhaften Verwendungen zu vermeiden. Es ist auch möglich, Empfehlungen auf spezifische Problematiken zu geben, um eine gute, zeitlose Qualität der Texte zu gewährleisten.

Die Schemasprache Schematron, die Nutzung von regulären Ausdrücken sowie die mögliche Ausgabe von Elementinhalten bzw. Zeichenketten in den Fehlermeldungstexten erwiesen sich – neben den Technologien für die finale Erstellung des Prüfschemas – auch als gute Tools zum Analysieren von den ONIX-Daten. Mit Schematron lassen sich auch kundenspezifische Regeln formulieren, um Verarbeitungsproblematiken vorzubeugen, was auch für die Qualität der einzelnen Arbeitsprozesse signifikant ist. Hier zeigt sich, dass das Schema dieser vorliegenden Arbeit zwar auf alle ONIX-Dateien anwendbar ist, aber auf die zwölf ONIX-Meldungen aus einer Auswahl von Verlagen bzw. Distributoren zugeschnitten ist. Es wurde nicht genau untersucht, vor welchem Arbeitsprozess es sich am meisten lohnt, die Schematron-Validierung durchzuführen, und es wurde nicht ausgearbeitet, welche Verlagssoftware oder Entwicklertools ONIX-Metadatenmanager nutzen. Inwiefern die Fehlerbehebung umgesetzt werden kann bzw. soll und wie ein geeignetes Fehlerreport-Dokument erzeugt werden kann, das auf die Bedürfnisse kleinerer Nutzergruppen zugeschnitten ist, könnte neben den eben genannten Punkten der Gegenstand neuer Untersuchungen in einer weiteren Arbeit sein. Auf die Thematik der Text-Format-Codes und der freien Textinhalte kann im Schematron-Schema dieser Arbeit in Zukunft noch tiefer eingegangen werden. So zeigte sich bei der Analyse der Metadaten mit dem Vergleich der entsprechenden Informationen auf Websites die Problematik mit einem reinen Textformat insbesondere bei Inhaltsverzeichnissen. Dies wurde zum Anlass genommen, eine Prüfregel zur problematischen Darstellung von TOCs in das Schema aufzunehmen. Es stellte sich heraus, dass die Konsistenz von der Angabe der Textformate und charakteristische Merkmale dieser aber auch überprüft werden könnte. Gleichzeitig könnte in weiteren Regeln die empfohlene Nutzung von XHTML innerhalb von ONIX-Elementen stärker in den Fokus rücken.

Um ein effizientes Schema zu formulieren, das auch externe Nutzer nutzen können, ist auf einen gewissen Grad der Selbstbeschreibungsfähigkeit zu achten. Dies wird erreicht durch eindeutige Benennungen in den Prüfregeln, die Nennung von sowohl Reference- als auch Short-Tag-Namen in den Fehlermeldungstexten und die einheitlich aufgebaute Mitgabe von Lösungsvorschlägen. Aus technologischer Sicht sind die Klassifizierung der Regeln in Phasen und die Nutzung von XSLT-Textausgaben geeignete Mittel, um die Benutzerfreundlichkeit zu erhöhen.

In Hinblick auf ONIX 3.0 zeigt sich, dass Zweckentfremdungen weiterhin auftreten und auch fast alle für ONIX 2.1 erstellten Prüfregeln in ONIX 3.0 weiterhin Anwendung finden, um Problematiken aufzuspüren. Dennoch gibt es in ONIX 3.0 eine weiterentwickelte Struktur mit vielen neuen ONIX-Elementen, die an den Stand der Technik und an die Veränderungen des Buchhandels angepasst ist. Da ONIX 2.1 nach wie vor viel sowohl im deutschsprachigen Raum als auch global verwendet wird und dies nach Schätzungen auch noch einige Jahre andauern wird, war es keine schlechte Entscheidung, die nicht-aktuelle ONIX-Version zu untersuchen und ein Schematron-Schema für sie zu erstellen. Dadurch, dass die Schematron-Regeln durch kleinere Anpassungen auch auf ONIX 3.0 anwendbar sind und auch dort Problematiken aufspüren, hat das Schema einen umso größeren Mehrwert. Die Prüfregeln bieten Grundlagen, um weitere, verwandte Regeln abzuleiten oder sie z. B. an eine andere Sprache anzupassen. Gleichzeitig bietet die Open-Source-Bereitstellung des Schematron-Schemas die Möglichkeit, dass Interessenten Anregungen geben können, welche dazu beitragen, das Schema weiterzuentwickeln. Dies könnte die Erstellung von sowohl allgemeinen Regeln als auch die Ausführung von kundenspezifischen Wünschen sein, die in Aufträgen für den Praxispartner dieser Arbeit resultieren können. Zudem kann die Sichtung der Prüfregeln und externes Feedback dazu helfen, die sinnvolle Nutzung der neuen ONIX-Elemente in ONIX 3.0 auszuschöpfen.

Literaturverzeichnis

BaseX Documentation (2022). Indexes. Online verfügbar unter https://docs.basex.org/wiki/Indexes#Path_Index (abgerufen am 30.11.2022).

Becher, Margit (2021). XML. DTD, XML-Schema, XPath, XQuery, XSL-FO, SAX, DOM. 2. Aufl. Wiesbaden, Springer Vieweg.

Bongers, Frank (2008). XSLT 2.0 & XPath 2.0. Das umfassende Handbuch. Grundlagen, Anwendung, Referenz, kommentierte Referenz aller Funktionen und Elemente, Installation und Anwendung von Saxon 9.0 und Altova XML 2008, auf CD: XML-/XSLT-Editoren. 2. Aufl. Bonn, Galileo Press.

data2type GmbH (2022). Gültige Dokumente korrekte Dokumente. Qualitätssicherung von XML-Dateien: DTD - Schema - Schematron – RelaxNG. Online verfügbar unter https://www.data2type.de/fileadmin/Vortraege/Qualitaetssicherung/Qualitaetssicherung.pdf (abgerufen am 05.12.2022).

EDItEUR (2006). ONIX for Books: Product Information Message Product Record Format. Release 2.1, revision 03. Online verfügbar unter https://www.editeur.org/files/ONIX%202.1/ONIX_for_Books_Release2-1_rev03_docs+codes_Issue_36.zip (abgerufen am 13.12.2022), führt zu Download.

EDItEUR (2009). Frequently Asked Questions about ONIX for Books. EDITEUR FAQ on ONIX 3.0. Online verfügbar unter https://www.editeur.org/files/ONIX%203/20110517%20Standard%20FAQ%20ii%20(ONIX%203).pdf (abgerufen am 13.12.2022), führt zu Download.

EDItEUR (2017). ONIX for Books: Codelists Issue 36. Online verfügbar unter https://www.editeur.org/files/ONIX%20for%20books%20-%20code%20lists/ONIX_BookProduct_Codelists_Issue_36.html (abgerufen am 13.12.2022)

EDItEUR (2021). ONIX for Books: Product Information Format Specification. Release 3.0 revision 8, June 2021. Online verfügbar unter https://www.editeur.org/files/ONIX%203/ONIX_for_Books_Release3-0_docs+codes_Issue_59.zip (abgerufen am 13.12.2022), führt zu Download.

EDItEUR (2021). ONIX for Books: Product Information Message. Application Note: Embedding HTML markup in ONIX 3.0 data elements. Online verfügbar unter https://www.editeur.org/files/ONIX%203/APPNOTE%20HTML%20markup%20in%20ONIX.pdf (abgerufen am 13.12.2022), führt zu Download.

EDItEUR (2022). ONIX FAQs. Frequently asked questions. Online verfügbar unter https://www.editeur.org/74/FAQs/ (abgerufen am 13.12.2022).

EDItEUR (2022). ONIX for Books: Codeslists Issue 59. Online verfügbar unter https://www.editeur.org/files/ONIX%20for%20books%20-%20code%20lists/ONIX_BookProduct_Codelists_Issue_59.html (abgerufen am 13.12.2022)

EDItEUR (2023). ONIX tagname converter. Online verfügbar unter https://www.editeur.org/files/ONIX%203/ONIX%20tagname%20converter%20v2.htm (abgerufen am 27.01.2023).

EDItEUR (2006). ONIX for Books: Product Information Message XML Message Specification. Release 2.1, revision 03 January 2006. Online verfügbar unter https://www.editeur.org/15/Archived-Previous-Releases/ (abgerufen am 13.12.2022)

EDItEUR/Bell, Graham (2012). ONIX for Books: Summary of changes for ONIX for Books 3.0 revision 1. Online verfügbar unter https://www.editeur.org/files/ONIX%203/Changes%20for%20ONIX%203.0.1.pdf (abgerufen am 13.12.2022), führt zu Download.

EDItEUR/Bell, Graham (2014). Using local DTD and XSD files after ONIX 2.1 sunset. Online verfügbar unter https://www.editeur.org/files/ONIX%202.1/ONIX_2.1_local_DTD_and_XSD_instructions.pdf (abgerufen am 13.12.2022), führt zu Download.

Friedl, Jeffrey E. F. (2009). Reguläre Ausdrücke. 3. Aufl. s.l., O'Reilly Verlag.

Grupe, Wilfried (2022). XML: Schwach strukturierte Dokumente. Online verfügbar unter https://www.wilfried-grupe.de/XML_Strukturalternativen1.html (abgerufen am 07.12.2022).

Hedler, Marko/Montero Pineda, Manuel/Kutscherauer, Nico (2011). Schematron. Effiziente XML Business Rules für XML-Dokumente. Heidelberg, dpunkt-Verl.

Heimann, Holger (2015). Metadaten verkaufen mehr Bücher. Börsenblatt vom 30.12.2015. Online verfügbar unter https://www.boersenblatt.net/archiv/1073508.html (abgerufen am 07.12.2022).

Herczeg, Michael (2018). Software-Ergonomie. Theorien, Modelle und Kriterien für gebrauchstaugliche interaktive Computersysteme. 4. Aufl. Berlin/Boston, De Gruyter Oldenbourg.

Herlinger, Barbara (2022). Zweckentfremdungen in ONIX 2.1 und 3.0. Interview durch Herta Albrecht am 10.11.2022.

Hiller, Simon (2016). Buchhandelsstrategien im digitalen Markt. Reaktionen der großen Buchhandelsketten auf technologische Neuerungen. Berlin/Boston, De Gruyter Saur.

HTML-Seminar.de (2023). Zeilenumbruch erzwingen über <br>. Online verfügbar unter https://www.html-seminar.de/html-umbruch.htm (abgerufen am 27.01.2023).

Jelliffe, Rick (2023). Schematron. Online verfügbar unter https://www.schematron.com/ (abgerufen am 22.01.2023).

Kutscherauer, Nico (2023). Schematron QuickFix project. Online verfügbar unter https://www.schematron-quickfix.com/ (abgerufen am 31.01.2023).

Lackes, Richard (2018). Definition: Business Rule. Springer Fachmedien Wiesbaden GmbH vom 19.02.2018. Online verfügbar unter https://wirtschaftslexikon.gabler.de/definition/business-rule-31273 (abgerufen am 29.12.2022).

Mangano, Sal (2006). XSLT-Kochbuch. Lösungen für XML- und XSLT-Entwickler. 2. Aufl. Köln, O'Reilly.

MVB GmbH (2022). Über das VLB. Online verfügbar unter https://vlb.de/ueber-uns/ueber-uns (abgerufen am 07.12.2022).

.NET (2022). Sprachelemente für reguläre Ausdrücke – Kurzübersicht. Microsoft. Online verfügbar unter https://learn.microsoft.com/de-de/dotnet/standard/base-types/regular-expression-language-quick-reference (abgerufen am 07.12.2022).

OASIS (2005). XML Catalogs. OASIS Standard V1.1, 7 October 2005. Online verfügbar unter https://www.oasis-open.org/committees/download.php/14810/xml-catalogs.pdf (abgerufen am 13.12.2022), führt zu Download.

ONIX Anwender-Gruppe für den deutschsprachigen Raum (2010). Best Practices ONIX for Books (Version 2.1). E-Book Standardmeldung. Ausgabe Oktober 2010. Online verfügbar unter https://home.bic-media.com/onix_info/best_practices_onix_for_books.pdf (abgerufen am 13.12.2022), führt zu Download.

Pantopix (2021). DTD & Schematron vs. XML Schema. Schemasprachen für die Technische Dokumentation. Online verfügbar unter https://www.pantopix.com/files/Whitepaper_Schemasprachen.pdf (abgerufen am 05.12.2022).

Pufe, Maren (2022). Wichtige ONIX-Elemente und Reklamationen. Interview durch Herta Albrecht am 2022.

Rühle, Stefanie (2012). Kleines Handbuch - Metadaten. Kompetenzzentrum Interoperable Metadaten. Online verfügbar unter https://wiki.dnb.de/download/attachments/43523047/201209_metadaten.pdf (abgerufen am 13.12.2022), führt zu Download.

Saxonica (2022). Technology. Online verfügbar unter https://www.saxonica.com/technology/technology.xml (abgerufen am 28.11.2022).

Siegel, Erik (2022). Schematron. A language for validating XML. Denver, XML PRESS.

Synchro Soft SRL (2020). Oxygen XML Editor 24.1. User Guide. Online verfügbar unter https://www.oxygenxml.com/xml_editor/software_archive_editor.html (abgerufen am 09.12.2022)

Voigt, Kai-Ingo (2022). Definition: Qualitätssicherung. Online verfügbar unter https://wirtschaftslexikon.gabler.de/definition/qualitaetssicherung-44396 (abgerufen am 07.12.2022).

Vonhoegen, Helmut (2018). XML. Einstieg, Praxis, Referenz. 9. Aufl. Bonn, Rheinwerk Verlag.

W3C (2012). Associating Schemas with XML documents 1.0 (Third Edition). Online verfügbar unter https://www.w3.org/TR/xml-model/ (abgerufen am 13.12.2022)

W3C (2022). The Extensible Stylesheet Language Family (XSL). Online verfügbar unter https://www.w3.org/Style/XSL/ (abgerufen am 28.11.2022).

W3Schools (2023). XPath, XQuery, and XSLT Function Reference. Online verfügbar unter https://www.w3schools.com/xml/xsl_functions.asp (abgerufen am 16.01.2023).

Walter, David (2016). Nielsen Book US Study: The Importance of Metadata for Discoverability and Sales.

Anhang A Schematron-Schema für ONIX 2.1 mit XSLT

Anmerkung: Die Verweise in eckigen Klammern befinden sich in XML-Kommentaren, um den Code zum Testen einfacher kopieren zu können.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Bachelorarbeit: Schematron-Schema für ONIX Version 2.1 -->
<schema xmlns="http://purl.oclc.org/dsdl/schematron"
queryBinding="xslt2"
  xmlns:sqf="http://www.schematron-quickfix.com/validator/process"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <!-- Prüfregeln -->

  <!-- FalscheVerwendung -->
  <pattern id="EditionsbeschreibungFehler">
    <rule role="error" context="b058">
      <assert
        test="matches(.,
'\d[.]|\d(st|nd|rd|th)|(erst|zweit|dritt|viert|fünft|sechst|siebt|acht|n
eunt|zehnt)', 'i')">
        Die Beschreibung in b058 (EditionStatement) ist möglicherweise
unvollständig oder fehlerhaft.
        Fundstelle(n): <xsl:value-of select="."/>
      </assert>
    </rule>
  </pattern>

  <pattern id="AbbildungsnotizZiffer">
    <rule role="error" context="b062">
      <report test=". castable as xs:integer">
        In b062 (IllustrationsNote) steht nur eine Nummer, obwohl hier
eine
        Anmerkung stehen soll. Vorschlag: Überführe die Zahl in b125
(NumberOfIllustrations).
      </report>
    </rule>
  </pattern>

  <pattern id="UntertitelAuflage">
    <rule role="error" context="b029">
      <report test="contains(., 'Auflage')"> In b029 (Subtitle) steht
die Auflage.
        Vorschlag: Überführe den Text in b058 (EditionStatement).
      </report>
    </rule>
  </pattern>

  <pattern id="AbbildungsnotizSeiten">
    <rule role="warning" context="b062">
      <report test="matches(., 'S[.]')">
        Befindet sich eine Seitenzahlangabe in b062 (IllustrationsNote)?
        Vorschlag: Überführe die Seitenzahlangabe in b061
(NumberOfPages).
      </report>
    </rule>
  </pattern>
```

```
  <pattern id="NonbookmeldungV21">
    <rule context="
        product/b012[. = 'BZ'] | product/b012[. = 'PZ'] |
        product/b012[. = 'ZA'] |
        product/b012[. = 'ZZ'] | product/b012[. = '00']">
      <report role="information"
        test="exists(.) and not(parent::product/title[matches(.,
'spiel|quiz|game', 'i')])">
        Es handelt sich möglicherweise um ein Nonbookprodukt, das
ungenau kategorisiert wurde.
        Vorschlag: Passe in b012 (ProductForm) den Code an.
      </report>
      <report role="warning"
        test="parent::product/title[matches(., 'spiel|quiz|game',
'i')]">
        Handelt es sich bei dem Produkt um ein Spiel?
        Vorschlag: In b012 (ProductForm) den Code ZE nutzen.
      </report>
    </rule>
  </pattern>

  <pattern id="UntertitelErweiterung">
    <rule role="warning" context="b029">
      <report test="contains(., 'Book')">
        In b029 (Subtitle) stehen möglicherweise Marketinginformationen.
        Vorschlag: Überführe die Information in ein Other text
composite.
      </report>
    </rule>
  </pattern>

  <!-- BiografieFehler -->
  <pattern id="ContributorGesamtbio">
    <rule role="error" context="
        product[count(contributor) &gt; 1]
        [contributor/b044]
        /othertext[d102 = '13']/d104">
      <report test="ancestor::product[1]/contributor[b044]/b044
                    /normalize-space() = normalize-space(.)">
        Der Biografietext eines Contributors stimmt mit dem Text für
alle Contributoren im Other text composite überein,
        obwohl es darin aber um mehrere Contributoren gehen sollte.
        Vorschlag: Der Biografietext im Other text composite sollte
gelöscht werden.
      </report>
    </rule>
  </pattern>

  <pattern id="EinContributorBios">
    <rule role="error" context="
        product[count(contributor) = 1]
        [contributor/b044]
        /othertext[d102 = '13']/d104">
      <assert test="ancestor::product[1]/contributor[b044]/b044
                    /normalize-space() = normalize-space(.)">
        Es gibt nur einen Contributor und dessen Biografietext weicht
vom Text für alle Contributoren ab. Wenn es nur einen
        Contributor gibt, benötigt es keine Gesamtbiografie im Other
text composite.
      </assert>
    </rule>
  </pattern>
```

```
  <pattern id="BioName">
    <rule role="error" context="b044" >
      <let name="b036"
        value="ancestor::contributor/b036/normalize-space()"/>
      <assert test="matches(., $b036)">
        Der Name des Contributors muss im Biografietext vorkommen und
sollte mit dem angegebenen Namen ('<xsl:value-of select="$b036"/>')
        übereinstimmen.
      </assert>
    </rule>
  </pattern>

  <pattern id="BioURL">
    <rule role="error" context="b044">
      <report test="contains(., 'www.') or contains(., '.de')
            or contains(., '.com')">
        Eine URL sollte nicht in einer Biographical note stehen.
        Vorschlag: Nutze dafür das Website composite.
      </report>
    </rule>
  </pattern>

  <pattern id="AutorenBios">
    <rule role="warning" context="
        product[count(./contributor[b035 = 'A01']) &gt; 1]
        /contributor[b035 = 'A01']/b044">
      <report test="ancestor::product[1]
            [count(./contributor[b035 = 'A01'])
            != count(./contributor[b035 = 'A01']/b044)]">
        Es gibt ungleich viele Autoren und Biografien in b044
(BiographicalNote). Dies kann zu Fehlern in der Gesamtbiografie im Other
text composite führen.
      </report>
    </rule>
  </pattern>

  <!-- TextFehler -->
  <pattern id="FehlendesSpace">                                    <!--[7]-->
    <rule role="warning" context="d104 | b044">
      <let name="Punkt-Regex" value="'\w*\p{Ll}[.:!?]\p{Lu}\w*'"/>
      <xsl:variable name="VPunkt" as="xs:string*">                 <!--[1]-->
        <xsl:analyze-string select="." regex="{$Punkt-Regex}"><!--[2]-->
          <xsl:matching-substring>                                 <!--[3]-->
            <xsl:if test="not(. = 'Ph.D') and not(. = 'e.V') and
                          not(. = 'a.D') and not(. = 'z.B')">
              <xsl:value-of select="."/>
            </xsl:if>
          </xsl:matching-substring>
        </xsl:analyze-string>
      </xsl:variable>                                              <!--[4]-->
      <report test="exists($VPunkt)"> Fehlt ein Leerzeichen zwischen
zwei Sätzen?
        Fundstelle(n): <xsl:value-of select="string-join($VPunkt, ',
')"/>
      </report>
    </rule>
  </pattern>

  <pattern id="ZahlFehler">                                        <!--[8]-->
    <rule role="warning" context="d104">
      <let name="ZahlFehler-Regex" value="'\d+[?]\d+'"/>
```

```
      <report test="matches(., $ZahlFehler-Regex)">          <!--[5]-->
        <xsl:variable name="VZahlFehler" as="xs:string*">
          <xsl:analyze-string select="." regex="{$ZahlFehler-Regex}">
            <xsl:matching-substring>
              <xsl:value-of select="."/>
            </xsl:matching-substring>
          </xsl:analyze-string>
        </xsl:variable> In einer Zahl liegt möglicherweise ein Fehler
vor.
        Fundstelle(n): <xsl:value-of select="string-join($VZahlFehler,
', ')"/>                                                  <!--[6]-->
      </report>
    </rule>
  </pattern>

  <pattern id="UmlautFehler">                              <!--[9]-->
    <rule role="warning" context="contributor | d104">
      <let name="UmlautFehler-Regex" value="'\w*[aouAOU][?][a-z]\w*'"/>
      <report test="matches(., $UmlautFehler-Regex)">
        <xsl:variable name="VUmlautFehler" as="xs:string*">
          <xsl:analyze-string select="." regex="{$UmlautFehler-Regex}">
            <xsl:matching-substring>
              <xsl:value-of select="."/>
            </xsl:matching-substring>
          </xsl:analyze-string>
        </xsl:variable> Möglicherweise liegt ein Umlautfehler vor.
        Fundstelle(n): <xsl:value-of select="string-join($VUmlautFehler,
', ')"/>
      </report>
    </rule>
  </pattern>

  <pattern id="SonstigeKodierungsfehler">                  <!--[10]-->
    <rule role="warning" context="d104 | b044">

      <!-- A: Bsp.: 2018 ? 1 oder der ?Hirnstamm -->
      <let name="SpaceFragezeichen-Regex" value="'\w*\s[?]\W\w*'"/>
      <report test="matches(., $SpaceFragezeichen-Regex)">
        <xsl:variable name="VSpaceFragezeichen" as="xs:string*">
          <xsl:analyze-string select="." regex="{$SpaceFragezeichen-
Regex}">
            <xsl:matching-substring>
              <xsl:value-of select="."/>
            </xsl:matching-substring>
          </xsl:analyze-string>
        </xsl:variable> A: Hier liegen möglicherweise Zeichenfehler mit
einem Whitespace vor.
        Fundstelle(n): <xsl:value-of select="string-
join($VSpaceFragezeichen, ', ')"/>
      </report>

      <!-- B: Bsp.: 113?ff -->
      <let name="SonstigeKodierungsfehler1-Regex"
value="'\w*.[^a^o^u^A^O^U][?]\p{Ll}+'"/>
      <report test="matches(., $SonstigeKodierungsfehler1-Regex)">
        <xsl:variable name="VwkF1" as="xs:string*">
          <xsl:analyze-string select="."
regex="{$SonstigeKodierungsfehler1-Regex}" flags="s">
            <xsl:matching-substring>
              <xsl:value-of select="."/>
            </xsl:matching-substring>
          </xsl:analyze-string>
```

```
        </xsl:variable> B: Hier liegen möglicherweise Kodierungsfehler
vor.
        Fundstelle(n): <xsl:value-of select="string-join($VwkF1, ',
')"/>
      </report>

      <!-- C: Bsp.: St.?Gallen -->
      <let name="SonstigeKodierungsfehler2-Regex"
value="'\w*\.[?]\p{Lu}+'"/>
      <report test="matches(., $SonstigeKodierungsfehler2-Regex)">
        <xsl:variable name="VwkF2" as="xs:string*">
          <xsl:analyze-string select="."
regex="{$SonstigeKodierungsfehler2-Regex}" flags="s">
            <xsl:matching-substring>
              <xsl:value-of select="."/>
            </xsl:matching-substring>
          </xsl:analyze-string>
        </xsl:variable> C: Hier liegen möglicherweise Kodierungsfehler
vor.
        Fundstelle(n): <xsl:value-of select="string-join($VwkF2, ',
')"/>
      </report>

      <!-- D: Bsp.: §?175 oder kann?10 oder S.?75 -->
      <let name="SonstigeKodierungsfehler3-Regex" value="'.?\D[?]\d+'"/>
      <report test="matches(., $SonstigeKodierungsfehler3-Regex)">
        <xsl:variable name="VwkF3" as="xs:string*">
          <xsl:analyze-string select="."
regex="{$SonstigeKodierungsfehler3-Regex}" flags="s">
            <xsl:matching-substring>
              <xsl:value-of select="."/>
            </xsl:matching-substring>
          </xsl:analyze-string>
        </xsl:variable> D: Hier liegen möglicherweise Kodierungsfehler
vor.
        Fundstelle(n): <xsl:value-of select="string-join($VwkF3, ',
')"/>
      </report>

    </rule>
  </pattern>

  <!-- Empfehlung -->
  <pattern id="SeitJahren">
    <rule role="warning"
      context="b044 | othertext[d102 = '13']/d104">
      <report
        test="matches(.,
'seit[\p{Zs}\s]+(\d+|zwei|drei|vier|fünf|sechs|sieben|acht|neun|zehn|elf
|zwölf)[\p{Zs}\s]+Jahren', 'i')">
        Die Information in der Biografie kann veraltet sein.
        Vorschlag: "seit dem Jahr ..." oder " seit über ... Jahren".
      </report>
    </rule>
  </pattern>

  <pattern id="KeywordInflation">
    <rule role="information" context="product/subject[1]">
      <report test="ancestor::product[count(./b067/'20') &gt; 35]">
Reduziere die Keywords. Es sind über 35 Keywords zu
        diesem Produkt eingetragen.
      </report>
```

```
    </rule>
  </pattern>

  <pattern id="Zeichenlaenge"><!-- individuelle Empfehlungen oder
Wünsche auch mgl. -->
    <rule role="information" context="b203">
      <report test="./string-length() &gt; 300">
        Die empfohlene Länge beträgt hier 300 Zeichen. Wenn möglich,
kürze den Text.
      </report>
    </rule>
  </pattern>

  <pattern id="TOCDefaultTextFormat">
    <rule role="information" context="othertext[d102 = '04']
                                      [d103 = '06']/d104      |
                      othertext[d102='04'][@textformat='06']">
      <report test="not(matches(., 'br\s*/'))">
        Das TOC könnte auf eine Website übernommen werden und so seine
Struktur verlieren.
        Vorschlag: Nutze bei d103 den Code 05 und überführe das TOC in
eine XHTML-Struktur.
      </report>
    </rule>
  </pattern>
</schema>
```

Listing A.1: Schematron-Schema für ONIX 2.1 mit XSLT[204]

[204] Auch zu finden unter
https://github.com/transpect/schema-onix/blob/master/Schematron/onix_21.sch

Anhang B Schematron-Schema für ONIX 2.1 und 3.0 mit Phasen und XSLT

Anmerkung: Die Verweise in eckigen Klammern befinden sich in XML-Kommentaren, um den Code zum Testen einfacher kopieren zu können.

```
<?xml version="1.0" encoding="UTF-8"?>
<!-- Bachelorarbeit: Schematron-Schema für ONIX 2.1 und 3.0 -->
<schema xmlns="http://purl.oclc.org/dsdl/schematron"
queryBinding="xslt2"
  xmlns:sqf="http://www.schematron-quickfix.com/validator/process"
xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <phase id="EinzelTest">                                    <!-- [1] -->
<!--Für Anpassung dem Attributwert die ID des entspr. Patterns geben-->
    <active pattern="NonbookmeldungV21"/>
  </phase>
  <phase id="FalscheVerwendung">
    <active pattern="EditionsbeschreibungFehler"/>
    <active pattern="AbbildungsnotizZiffer"/>
    <active pattern="UntertitelAuflage"/>
    <active pattern="AbbildungsnotizSeiten"/>
    <active pattern="NonbookmeldungV21"/>
    <active pattern="NonbookmeldungV30"/>
    <active pattern="UntertitelErweiterung"/>
  </phase>
  <phase id="BiografieFehler">
    <active pattern="ContributorGesamtbio"/>
    <active pattern="EinContributorBios"/>
    <active pattern="BioName"/>
    <active pattern="BioURL"/>
    <active pattern="AutorenBios"/>
  </phase>
  <phase id="TextFehler">
    <active pattern="FehlendesSpace"/>
    <active pattern="ZahlFehler"/>
    <active pattern="UmlautFehler"/>
    <active pattern="SonstigeKodierungsfehler"/>
<!--    <active pattern="Bindestrich"/>-->
<!--    <active pattern="DreiBuchstaben"/>-->
  </phase>
  <phase id="Empfehlung">
    <active pattern="SeitJahren"/>
    <active pattern="KeywordInflation"/>
    <active pattern="Zeichenlaenge"/>
    <active pattern="TOCDefaultTextFormat"/>
  </phase>

  <properties>                                               <!-- [3] -->
    <property id="refname"><xsl:value-of select="@refname"/></property>
  </properties>

  <!-- Prüfregeln -->                                        <!-- [2] -->

  <!-- FalscheVerwendung -->
  <pattern id="EditionsbeschreibungFehler">
    <rule role="error" context="*:b058">
      <assert
```

```
        test="matches(.,
'\d[.]|\d(st|nd|rd|th)|(erst|zweit|dritt|viert|fünft|sechst|siebt|acht|n
eunt|zehnt)', 'i')"
        properties="refname">
        Die Beschreibung in b058 (EditionStatement) ist möglicherweise
unvollständig oder fehlerhaft.
        Fundstelle(n): <xsl:value-of select="."/>
      </assert>
    </rule>
  </pattern>

  <pattern id="AbbildungsnotizZiffer">
    <rule role="error" context="*:b062">
      <report
        test=". castable as xs:integer"
        properties="refname">
        In b062 (IllustrationsNote) steht nur eine Nummer, obwohl hier
eine Anmerkung stehen soll.
        Vorschlag: Überführe die Zahl in b125 (NumberOfIllustrations).
      </report>
    </rule>
  </pattern>

  <pattern id="UntertitelAuflage">
    <rule role="error" context="*:b029">
      <report
        test="contains(., 'Auflage')"
        properties="refname">
        In b029 (Subtitle) steht die Auflage.
        Vorschlag: Überführe den Text in b058 (EditionStatement).
      </report>
    </rule>
  </pattern>

  <pattern id="AbbildungsnotizSeiten">
    <rule role="warning"
      context="*:b062">
      <report
        test="matches(., 'S[.]')"
        properties="refname" >
        Befindet sich eine Seitenzahlangabe in b062 (IllustrationsNote)?
Vorschlag: Überführe die Seitenzahlangabe in b061 (NumberOfPages).
      </report>
    </rule>
  </pattern>

<!--  NonbookmeldungV21 für ONIX 2.1 und NonbookmeldungV30 für ONIX 3.0-
->
  <pattern id="NonbookmeldungV21">                          <!-- [4] -->
    <rule context="product/b012[.='BZ'] | product/b012[.='PZ'] |
product/b012[.='ZA'] | product/b012[.='ZZ'] | product/b012[.='00']" >
      <report role="information"
        test="exists(.) and not(parent::product/title[matches(.,
'spiel|quiz|game', 'i')])"
        properties="refname">
        V2.1: Es handelt sich möglicherweise um ein Nonbookprodukt, das
ungenau kategorisiert wurde.
        Vorschlag: In b012 (ProductForm) den Code anpassen.
      </report>
      <report role="warning"
        test="parent::product/title[matches(., 'spiel|quiz|game', 'i')]"
```

```
        properties="refname">
        V2.1: Handelt es sich bei dem Produkt um ein Spiel? Vorschlag:
In b012 (ProductForm) den Code ZE nutzen.
      </report>                                                <!-- [6] -->
    </rule>
  </pattern>

  <pattern id="NonbookmeldungV30">                             <!-- [5] -->
    <rule context="*:product/*:descriptivedetail/*:b012[.='BZ'] |
*:product/*:descriptivedetail/*:b012[.='PZ'] |
*:product/*:descriptivedetail/*:b012[.='ZA'] |
      *:product/*:descriptivedetail/*:b012[.='ZZ'] |
*:product/*:descriptivedetail/*:b012[.='00']" >
      <report role="information"
        test="exists(.) and
not(parent::*:descriptivedetail/*:titledetail[matches(.,
'spiel|quiz|game', 'i')])"
        properties="refname">
        V3.0: Es handelt sich möglicherweise um ein Nonbookprodukt, das
ungenau kategorisiert wurde. Vorschlag: In b012 (ProductForm) den Code
anpassen.
      </report>
      <report role="warning"
        test="parent::*:descriptivedetail/*:titledetail[matches(.,
'spiel|quiz|game', 'i')]"
        properties="refname">
        V3.0: Handelt es sich bei dem Produkt um ein Spiel? Vorschlag:
In b012 (ProductForm) den Code C7 nutzen (siehe List 27).
      </report>                                                <!-- [7] -->
    </rule>
  </pattern>

  <pattern id="UntertitelErweiterung">
    <rule role="warning" context="*:b029">
      <report
        test="contains(., 'Book')"
        properties="refname">
        In b029 (Subtitle) stehen möglicherweise Marketinginformationen.
        Vorschlag: Überführe die Information in ein Other text bzw. Text
content composite.
      </report>
    </rule>
  </pattern>

  <!-- BiografieFehler -->

  <pattern id="ContributorGesamtbio">
    <rule role="error" context="
      *:product[count(*:contributor) &gt; 1]
      [*:contributor/*:b044]
      /*:othertext[*:d102 = '13']/*:d104 |

      *:product[count(.//*:contributor) &gt; 1]
      [.//*:contributor/*:b044]
      //*:textcontent[*:x426 = '12']/*:d104">
      <report
        test="ancestor::*:product[1]//*:contributor[*:b044]/*:b044
/normalize-space() = normalize-space(.)"
        properties="refname">
        Der Biografietext eines Contributors stimmt mit dem Text für
alle Contributoren im Other text bzw. Text content composite überein,
```

```
        obwohl es darin aber um mehrere Contributoren gehen sollte.
        Vorschlag: Der Biografietext im Other text bzw. Text content
composite sollte gelöscht werden.
      </report>
    </rule>
  </pattern>

  <pattern id="EinContributorBios">
    <rule role="error" context="
      *:product[count(*:contributor) = 1]
      [*:contributor/*:b044]
      /*:othertext[*:d102 = '13']/*:d104 |

      *:product[count(.//*:contributor) = 1]
      [.//*:contributor/*:b044]
      //*:textcontent[*:x426 = '12']/*:d104">
      <assert
        test="ancestor::*:product[1]//*:contributor[*:b044]/*:b044
/normalize-space() = normalize-space(.)"
        properties="refname">
        Es gibt nur einen Contributor und dessen Biografietext weicht
vom Text für alle Contributoren ab.
        Wenn es nur einen Contributor gibt, benötigt es keine
Gesamtbiografie im Other text bzw. Text content composite.
      </assert>
    </rule>
  </pattern>

  <pattern id="BioName">
    <rule role="error" context="*:b044" >
      <let name="b036"
           value="ancestor::*:contributor/*:b036/normalize-space()"/>
      <assert test="matches(., $b036)">
        Der Name des Contributors muss im Biografietext vorkommen und
sollte mit dem angegebenen Namen ('<xsl:value-of select="$b036"/>')
        übereinstimmen.
      </assert>
    </rule>
  </pattern>


  <pattern id="BioURL">
    <rule role="error" context="*:b044">
      <report
        test="contains(., 'www.') or contains(., '.de') or contains(.,
'.com')"
        properties="refname">
        Eine URL sollte nicht in einer Biographical note stehen.
        Vorschlag: Nutze dafür das Website composite.
      </report>
    </rule>
  </pattern>

  <pattern id="AutorenBios">
    <rule role="warning"
      context="*:product[count(.//*:contributor[*:b035 = 'A01']) &gt; 1]
      //*:contributor[b035 = 'A01']/*:b044 ">
      <report
        test="ancestor::*:product[1]
        [count(.//*:contributor[*:b035 = 'A01']) !=
count(.//*:contributor[*:b035 = 'A01']/*:b044)]"
        properties="refname">
```

```
        Es gibt ungleich viele Autoren und Biografien in b044
(BiographicalNote). Dies kann zu Fehlern in der Gesamtbiografie im Other
text bzw. Text content composite führen.
      </report>
    </rule>
  </pattern>

<!-- TextFehler -->

  <pattern id="FehlendesSpace">
    <rule role="warning" context="*:d104 | *:b044" >
      <let name="Punkt-Regex" value="'\w*\p{Ll}[.:!?]\p{Lu}\w*'"/>
      <xsl:variable name="VPunkt" as="xs:string*">
        <xsl:analyze-string select="." regex="{$Punkt-Regex}">
          <xsl:matching-substring>
            <xsl:if test="
              not(. = 'Ph.D')and
              not(.='e.V') and
              not(.='a.D') and
              not(.='z.B')">
              <xsl:value-of select="."/>
            </xsl:if>
          </xsl:matching-substring>
        </xsl:analyze-string>
      </xsl:variable>
      <report
        test="exists($VPunkt)"
        properties="refname">
        Fehlt ein Leerzeichen zwischen zwei Sätzen? Fundstelle(n):
<xsl:value-of select="string-join($VPunkt, ', ')"/>
      </report>
    </rule>
  </pattern>

  <pattern id="ZahlFehler">
    <rule role="warning"
      context="*:d104" >
      <let name="ZahlFehler-Regex" value="'\d+[?]\d+'"/>
      <report
        test="matches(., $ZahlFehler-Regex)"
        properties="refname">
        <xsl:variable name="VZahlFehler" as="xs:string*">
          <xsl:analyze-string select="." regex="{$ZahlFehler-Regex}">
            <xsl:matching-substring>
              <xsl:value-of select="."/>
            </xsl:matching-substring>
          </xsl:analyze-string>
        </xsl:variable>
        In einer Zahl liegt möglicherweise ein Fehler vor.
        Fundstelle(n): <xsl:value-of select="string-join($VZahlFehler,
', ')" />
      </report>
    </rule>
  </pattern>

  <pattern id="UmlautFehler">
    <rule role="warning"
      context="*:contributor | *:d104">
      <let name="UmlautFehler-Regex" value="'\w*[aouAOU][?][a-z]\w*'"/>
      <report
        test="matches(., $UmlautFehler-Regex)"
        properties="refname">
```

```
        <xsl:variable name="VUmlautFehler" as="xs:string*">
          <xsl:analyze-string select="." regex="{$UmlautFehler-Regex}">
            <xsl:matching-substring>
              <xsl:value-of select="."/>
            </xsl:matching-substring>
          </xsl:analyze-string>
        </xsl:variable>
        Möglicherweise liegt ein Umlautfehler vor.
        Fundstelle(n): <xsl:value-of select="string-join($VUmlautFehler,
', ')" />
      </report>
    </rule>
  </pattern>

  <pattern id="SonstigeKodierungsfehler">
    <rule role="warning"
      context="*:d104 | *:b044">
      <!-- A: Bsp.: 2018 ? 1 oder der ?Hirnstamm -->
      <let name="SpaceFragezeichen-Regex" value="'\w*\s[?]\W\w*'"/>
      <report
        test="matches(., $SpaceFragezeichen-Regex)"
        properties="refname">
        <xsl:variable name="VSpaceFragezeichen" as="xs:string*">
          <xsl:analyze-string select="." regex="{$SpaceFragezeichen-
Regex}">
            <xsl:matching-substring>
              <xsl:value-of select="."/>
            </xsl:matching-substring>
          </xsl:analyze-string>
        </xsl:variable>
        A: Hier liegen möglicherweise Zeichenfehler mit einem Whitespace
vor.
        Fundstelle(n): <xsl:value-of select="string-
join($VSpaceFragezeichen, ', ')" />
      </report>
      <!-- B: Bsp.: 113?ff -->
      <let name="SonstigeKodierungsfehler1-Regex"
value="'\w*.[^a^o^u^A^O^U][?]\p{Ll}+'"/>
      <report
        test="matches(., $SonstigeKodierungsfehler1-Regex)"
        properties="refname">
        <xsl:variable name="VwkF1" as="xs:string*">
          <xsl:analyze-string select="."
regex="{$SonstigeKodierungsfehler1-Regex}" flags="s">
            <xsl:matching-substring>
              <xsl:value-of select="."/>
            </xsl:matching-substring>
          </xsl:analyze-string>
        </xsl:variable>
        B: Hier liegen möglicherweise Kodierungsfehler vor.
Fundstelle(n): <xsl:value-of select="string-join($VwkF1, ', ')" />
      </report>
      <!-- C: Bsp.: St.?Gallen -->
      <let name="SonstigeKodierungsfehler2-Regex"
value="'\w*\.[?]\p{Lu}+'"/>
      <report
        test="matches(., $SonstigeKodierungsfehler2-Regex)"
        properties="refname">
        <xsl:variable name="VwkF2" as="xs:string*">
          <xsl:analyze-string select="."
regex="{$SonstigeKodierungsfehler2-Regex}" flags="s" >
            <xsl:matching-substring>
```

```
              <xsl:value-of select="."/>
            </xsl:matching-substring>
          </xsl:analyze-string>
        </xsl:variable>
        C: Hier liegen möglicherweise Kodierungsfehler vor.
Fundstelle(n): <xsl:value-of select="string-join($VwkF2, ', ')" />
      </report>
      <!-- D: Bsp.: §?175 oder kann?10 oder S.?75 -->
      <let name="SonstigeKodierungsfehler3-Regex" value="'.?\D[?]\d+'"/>
      <report
        test="matches(., $SonstigeKodierungsfehler3-Regex)"
        properties="refname">
        <xsl:variable name="VwkF3" as="xs:string*">
          <xsl:analyze-string select="."
regex="{$SonstigeKodierungsfehler3-Regex}" flags="s">
            <xsl:matching-substring>
              <xsl:value-of select="."/>
            </xsl:matching-substring>
          </xsl:analyze-string>
        </xsl:variable>
        D: Hier liegen möglicherweise Kodierungsfehler vor.
Fundstelle(n): <xsl:value-of select="string-join($VwkF3, ', ')" />
      </report>
    </rule>
  </pattern>

  <!-- Empfehlung -->

  <pattern id="SeitJahren">
    <rule role="warning" context="*:b044 | *:othertext[*:d102 =
'13']/*:d104 |
      *:textcontent[*:x426='12']/*:d104">
      <report
        test="matches(.,
'seit[\p{Zs}\s]+(\d+|zwei|drei|vier|fünf|sechs|sieben|acht|neun|zehn|elf
|zwölf)[\p{Zs}\s]+Jahren', 'i')"
        properties="refname">
        Die Information in der Biografie kann veraltet sein. Vorschlag:
"seit dem Jahr ..." oder " seit über ... Jahren".
      </report>
    </rule>
  </pattern>

  <pattern id="KeywordInflation">
    <rule role="information"
      context="*:product//*:subject[1]">
      <report
        test="ancestor::*:product[count(.//*:b067/'20') &gt; 35]"
        properties="refname">
        Reduziere die Keywords. Es sind über 35 Keywords zu diesem
Produkt eingetragen.
      </report>
    </rule>
  </pattern>

  <pattern id="Zeichenlaenge"> <!-- individuelle Empfehlungen oder
Wünsche möglich -->
    <rule role="information" context="*:b203">
      <report
        test="./string-length() &gt; 300"
        properties="refname">
```

```
        Die empfohlene Länge beträgt hier 300 Zeichen. Wenn möglich,
kürze den Text.
      </report>
    </rule>
  </pattern>

  <pattern id="TOCDefaultTextFormat">
<rule role="information"
      context="*:othertext[*:d102 = '04'][*:d103 = '06']/*:d104  |
      *:othertext[*:d102='04'][@textformat='06'] |

      *:textcontent[*:x426 = '04'][@textformat='06']">
      <report
        test="not(contains(., 'br\s*/'))"
        properties="refname">
        Das TOC könnte auf eine Website übernommen werden und so seine
Struktur verlieren.
        Vorschlag: V2.1: Nutze bei d103 den Code 05 und überführe das
TOC in eine XHTML-Struktur. V3.0: Nutze bei @textformat den Code 05 und
überführe das TOC in eine XHTML-Struktur.
      </report>
    </rule>
  </pattern>
</schema>
```

Listing B.1: Finales Schematron-Schema für ONIX 2.1 und 3.0[205]

[205] Auch zu finden unter https://github.com/transpect/schema-onix/blob/master/Schematron/onix_de.sch

Anhang C ONIX-2.1-Beispieldatei

Anmerkung: Das Sternchen im XML-Kommentar (`<!-- * -->`) bedeutet, dass hier eine Problematik eingefügt wurde.

```
<?xml version="1.0" encoding="UTF-8" standalone="yes"?>
<!-- Bachelorarbeit: ONIX 2.1 Musterdatei mit fuenf eingebauten Fehlern -->
<!-- Gegebenenfalls die DTD deklarieren -->
<!-- Schematron-Schema zuweisen -->
<ONIXmessage>
  <header>
    <m174>Testverlag</m174>
    <m175>Max Mustermann</m175>
    <m283>Mustermann@testverlag.de</m283>
    <m182>20140828</m182>
  </header>
  <product>
    <a001>9783593422336</a001>
    <a002>03</a002>
    <productidentifier>
      <b221>03</b221>
      <b244>9783593422336</b244>
    </productidentifier>
    <productidentifier>
      <b221>15</b221>
      <b244>9783593422336</b244>
    </productidentifier>
    <b012>DH</b012>
    <b211>002</b211>
    <title>
      <b202>01</b202>
      <b203>Die Macht der Liebe</b203>
      <b029>ein neuer Blick auf das größte Gefühl</b029>
    </title>
    <contributor>
      <b034>1</b034>
      <b035>A01</b035>
      <b036>Barbara Fredrickson</b036>
      <b039>Barbara</b039>
      <b040>Fredrickson</b040>
      <b044>Sie ist eine erfolgreiche Autorin.</b044><!-- * -->
    </contributor>
    <contributor>
      <b034>2</b034>
      <b035>B06</b035>
      <b036>Nicole Hölsken</b036>
      <b039>Nicole</b039>
      <b040>Hölsken</b040>
    </contributor>
    <b056>NED</b056>
    <b057>1</b057>
    <b058>1. Aufl.</b058>
    <language>
      <b253>01</b253>
      <b252>ger</b252>
    </language>
    <extent>
      <b218>22</b218>
      <b219>6.34</b219>
      <b220>19</b220>
    </extent>
    <mainsubject>
      <b191>26</b191>
      <b068>2.0</b068>
      <b069>150</b069>
    </mainsubject>
```

```
    <subject>
      <b067>23</b067>
      <b069>1790</b069>
      <b070>Psychologie / Modernes Leben</b070>
    </subject>
    <othertext>
      <d102>13</d102>
      <d103>06</d103>
      <d104>Dies ist ein Platzhaltertext für die Gesamtbiografie. Hier sollen
biografische Informationen u?ber die Autorin und die U?bersetzerin
stehen.</d104><!-- * -->
    </othertext>
    <othertext>
      <d102>04</d102>
      <d103>06</d103><!-- * -->
      <d104>Inhalt
        Einleitung
        1. Kapitel 8
        2. Kapitel 16
        3. Kapitel
        ...
     </d104>
    </othertext>
    <productwebsite>
      <b367>01</b367>
      <f123>http://www.verlag-muster.de/home/</f123>
    </productwebsite>
    <publisher>
      <b291>01</b291>
      <b081>Muster Verlag</b081>
    </publisher>
    <b209>Frankfurt am Main</b209>
    <b003>2014</b003>
    <relatedproduct>
      <h208>13</h208>
      <productidentifier>
        <b221>15</b221>
        <b244>9783593500027</b244>
      </productidentifier>
      <b012>BA</b012>
    </relatedproduct>
  </product>

  <product>
    <a001>9783862741427</a001>
    <a002>03</a002>
    <productidentifier>
      <b221>15</b221>
      <b244>9783862741427</b244>
    </productidentifier>
    <b012>DH</b012>
    <b211>029</b211>
    <set>
      <b023>Die Tribute von Panem</b023>
      <b026>3</b026>
      <b281>Flammender Zorn</b281>
    </set>
    <title>
      <b202>01</b202>
      <b203>Flammender Zorn</b203>
      <b029>Roman</b029>
    </title>
    <contributor>
      <b034>1</b034>
      <b035>A01</b035>
      <b036>Suzanne Collins</b036>
      <b039>Suzanne</b039>
      <b040>Collins</b040>
    </contributor>
```

```
    <b058>2023</b058><!-- * -->
    <language>
      <b253>01</b253>
      <b252>ger</b252>
    </language>
    <mainsubject>
      <b191>26</b191>
      <b068>2.0</b068>
      <b069>K</b069>
    </mainsubject>
    <productwebsite>
      <b367>01</b367>
      <f123>http://www.DieTributevonPanem.de</f123>
    </productwebsite>
    <publisher>
      <b291>01</b291>
      <b081>Verlag Friedrich Muster</b081>
    </publisher>
    <b209>Hamburg</b209>
    <b003>2011</b003>
  </product>

  <product>
    <a001>9873862712345</a001>
    <a002>03</a002>
    <productidentifier>
      <b221>01</b221>
      <b233>Publishers Order No</b233>
      <b244>64412</b244>
    </productidentifier>
    <productidentifier>
      <b221>02</b221>
      <b244>1234544371</b244>
    </productidentifier>
    <productidentifier>
      <b221>03</b221>
      <b244>9787654644123</b244>
    </productidentifier>
    <productidentifier>
      <b221>15</b221>
      <b244>9787654644123</b244>
    </productidentifier>
    <b012>BZ</b012><!-- * -->
    <b014>50 Quizfragen in einer Schachtel</b014>
    <productclassification>
      <b274>04</b274>
      <b275>49012345</b275>
    </productclassification>
    <title>
      <b202>01</b202>
      <b203>Schachtelspiel</b203>
      <b029>Unsere wunderbare Welt - Das Quiz</b029>
    </title>
    <contributor>
      <b034>1</b034>
      <b035>A01</b035>
      <b340>1</b340>
      <b036>Maria Musterfrau</b036>
      <b037>Musterfrau, Maria</b037>
      <b039>Maria</b039>
      <b040>Musterfrau</b040>
    </contributor>
    <b057>1</b057>
    <language>
      <b253>01</b253>
      <b252>ger</b252>
    </language>
    <b255>82</b255>
    <extent>
```

```
      <b218>00</b218>
      <b219>82</b219>
      <b220>03</b220>
    </extent>
    <mainsubject>
      <b191>26</b191>
      <b068>2.0</b068>
      <b069>1415</b069>
      <b070>Hardcover, Softcover, Hobby, Spielen, Raten</b070>
    </mainsubject>
    <subject>
      <b067>20</b067>
      <b070>Leben</b070>
    </subject>
    <subject>
      <b067>20</b067>
      <b070>Geschenk</b070>
    </subject>
    <subject>
      <b067>20</b067>
      <b070>Geschenkidee</b070>
    </subject>
    <subject>
      <b067>20</b067>
      <b070>Rätselspiel Kinder</b070>
    </subject>
    <subject>
      <b067>20</b067>
      <b070>Welt</b070>
    </subject>
    <subject>
      <b067>20</b067>
      <b070>Rätsel</b070>
    </subject>
    <subject>
      <b067>20</b067>
      <b070>Wunder</b070>
    </subject>
    <b073>02</b073>
    <audiencerange>
      <b074>18</b074>
      <b075>03</b075>
      <b076>8</b076>
    </audiencerange>
    <othertext>
      <d102>99</d102>
      <d104>CN</d104>
    </othertext>
    <publisher>
      <b291>01</b291>
      <b241>05</b241>
      <b243>Musterverlag</b243>
      <b081>Verlag</b081>
    </publisher>
    <b083>DE</b083>
    <b003>202301</b003>
    <measure>
      <c093>01</c093>
      <c094>10</c094>
      <c095>cm</c095>
    </measure>
    <measure>
      <c093>02</c093>
      <c094>12</c094>
      <c095>cm</c095>
    </measure>
    <measure>
      <c093>08</c093>
      <c094>200</c094>
```

```
      <c095>gr</c095>
    </measure>
  </product>
</ONIXmessage>
```

Listing C.1: ONIX-2.1-Beispieldatei mit eingebauten Fehlern

Anhang D XSLT-Pipeline zur Erzeugung eines HTML-Reports

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:xso="http://transpect.io/generate-xsl"
  xmlns:sch="http://purl.oclc.org/dsdl/schematron"
  xmlns:map="http://www.w3.org/2005/xpath-functions/map"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="3.0">

  <!-- Invocation: saxon -s:file:///path/to/doc.xml -xsl:normalize-
validate.xsl \
                    schema-uri=file:///path/to/schema.sch -o:out.html
-->

  <xsl:param name="schema-uri" as="xs:string" select="resolve-
uri('../schema/onix/Schematron/BA-onix_V21.sch', static-base-
uri())"/>
  <xsl:param name="debug" as="xs:boolean" select="false()"/>
  <xsl:param name="debug-dir-uri" as="xs:string?"/>

  <xsl:output name="debug" indent="true" omit-xml-
declaration="false"/>

  <xsl:variable name="fixes" as="document-node(element())"
    select="transform(map{'stylesheet-location': 'make-dtd-
valid.xsl',
                          'source-node': /*,
                          'base-output-uri': base-uri(/*)
                         }) => map:get(base-uri(/*))">
  </xsl:variable>

  <xsl:variable name="doctype" as="xs:string"
    select="if (/*/@release = '3.0')
            then if (/ONIXMessage)
                 then
'https://www.editeur.org/onix/3.0/reference/ONIX_BookProduct_3.0_refe
rence.dtd'
                 else
'https://www.editeur.org/onix/3.0/short/ONIX_BookProduct_3.0_short.dt
d'
            else if (/ONIXMessage)
                 then
'http://www.editeur.org/onix/2.1/reference/onix-international.dtd'
                 else 'http://www.editeur.org/onix/2.1/short/onix-
international.dtd'"/>

  <xsl:variable name="reserialized" as="xs:string"
    select="$fixes => serialize(map{'doctype-system': $doctype})"/>

  <xsl:variable name="shorttags" as="item()*"
    select="if (name($fixes/*) = 'ONIXMessage')
            then  map:get(transform(map{'stylesheet-location':
'../tagname-converter/switch-onix-tagnames-2.0.xsl',
                                        'source-node': $reserialized
=> parse-xml(),
                                        'base-output-uri': base-
uri(/*),
```

```
                                          'stylesheet-params':
map{xs:QName('result-document'): 'file:/foo/bar'}
                                        }), 'file:/foo/bar')
            else $reserialized => parse-xml()">
  </xsl:variable>

  <xsl:variable name="result" as="map(xs:string, item()*)"
    select="transform(map{'stylesheet-location': '../oxy-
schematron/xsl/validate-and-render-svrl.xsl',
                                          'source-node': $shorttags,
                                          'base-output-uri': base-
uri(/*),
                                          'stylesheet-params': map{

xs:QName('schema-uri'): $schema-uri,

xs:QName('full-path-notation'): '2'
                                                                    }
                                        })"/>

  <xsl:variable name="svrl-storage" as="xs:string"
select="replace(base-uri(/*), 'xml$', 'svrl')"/>
  <xsl:variable name="shorttags-storage" as="xs:string"
select="replace(base-uri(/*), 'xml$', 'short.xml')"/>

  <xsl:template match="/">
    <xsl:sequence select="map:get($result, base-uri(/*))"/>
    <xsl:result-document href="{$shorttags-storage}">
        <xsl:sequence select="$shorttags"/>
      </xsl:result-document>
    <xsl:for-each select="map:keys($result)[ends-with(., '.val')]">
      <xsl:result-document href="{$svrl-storage}">
        <xsl:sequence select="map:get($result, .)"/>
      </xsl:result-document>
    </xsl:for-each>

  </xsl:template>

</xsl:stylesheet>
```

Listing D.1: XSLT-Pipeline normalize-validate.xsl

Anhang E Short- und Reference-Tags

Short-Tag-Name	Reference-Tag-Name
<alternativename>*	<AlternativeName>
<b012>	<ProductForm>
<b029>	<Subtitle>
<b035>	<ContributorRole>
<b036>	<PersonName>
<b044>	<BiographicalNote>
<b058>	<EditionStatement>
<b062>	<IllustrationsNote>
<b067>	<SubjectSchemeIdentifier>
<b069>	<SubjectCode>
<b070>	<SubjectHeadingText>
<b203>	<TitleText>
<b336>	<ProductFormFeatureDescription>
<contributor>	<Contributor>
<d102>	<TextTypeCode>
<d103>	<TextFormat>
<d104>	<Text>
<othertext>	<OtherText>
<product>	<Product>
<subject>	<Subject>
<textcontent>*	<TextContent>
<title>	<Title>
<x414>*	<NameType>
<x426>*	<TextType>
<x427>*	<ContentAudience>

Tabelle E.1: Gegenüberstellung von Short-Tag- und Reference-Tag-Namen (Übersicht)

* Diese Elemente sind neu in ONIX 3.0.

Eidesstattliche Erklärung

Hiermit erkläre ich, dass ich die vorliegende Bachelorarbeit mit dem Titel *Qualitätssicherung von Buchmetadaten in ONIX for Books mit Schematron* selbständig verfasst habe, dass ich sie zuvor an keiner anderen Hochschule und in keinem anderen Studiengang als Prüfungsleistung eingereicht habe und dass ich keine anderen als die angegebenen Quellen und Hilfsmittel benutzt habe. Alle Stellen der Arbeit, die wörtlich oder sinngemäß aus Veröffentlichungen oder aus anderweitigen fremden Äußerungen entnommen wurden, sind als solche kenntlich gemacht.

Leipzig, 19.02.2023, Herta Albrecht