Metadatenmanagement in Bibliotheken mit KNIME und Catmandu

- überarbeitete Fassung -

Bachelorarbeit

im Studiengang
Bibliotheks- und Informationsmanagement

vorgelegt von

Fiona Zurek
Matr.-Nr.: 29905

am 29. November 2019
an der Hochschule der Medien Stuttgart

zur Erlangung des akademischen Grades eines Bachelor of Arts

Erstprüfer:	Prof. Magnus Pfeffer
Zweitprüferin:	Prof. Heidrun Wiesenmüller

Ehrenwörtliche Erklärung

„Hiermit versichere ich, Fiona Hanna Zurek, ehrenwörtlich, dass ich die vorliegende Bachelorarbeit mit dem Titel: „Metadatenmanagement in Bibliotheken mit KNIME und Catmandu“ selbstständig und ohne fremde Hilfe verfasst und keine anderen als die angegebenen Hilfsmittel benutzt habe. Die Stellen der Arbeit, die dem Wortlaut oder dem Sinn nach anderen Werken entnommen wurden, sind in jedem Fall unter Angabe der Quelle kenntlich gemacht. Die Arbeit ist noch nicht veröffentlicht oder in anderer Form als Prüfungsleistung vorgelegt worden.

Ich habe die Bedeutung der ehrenwörtlichen Versicherung und die prüfungsrechtlichen Folgen (§ 26 Abs. 2 Bachelor-SPO (6 Semester), § 24 Abs. 2 Bachelor-SPO (7 Semester), § 23 Abs. 2 Master-SPO (3 Semester) bzw. § 19 Abs. 2 Master-SPO (4 Semester und berufsbegleitend) der HdM) einer unrichtigen oder unvollständigen ehrenwörtlichen Versicherung zur Kenntnis genommen.“

Stuttgart, den 29. November 2019

Kurzfassung

Die vorliegende Arbeit beschäftigt sich mit dem Metadatenmanagement in Bibliotheken. Es wird untersucht, inwiefern die Programme KNIME und Catmandu geeignet sind, Bibliotheken bei typischen Aufgaben des Metadatenmanagements zu unterstützen. Die technischen Entwicklungen im Bereich Metadaten sind aufgrund der Vielzahl an Formaten, Schnittstellen und Anwendungen komplexer geworden. Um die Metadaten entsprechend aufbereiten und nutzen zu können, werden Informationen über die Eignung verschiedener Programme benötigt. KNIME und Catmandu werden sowohl theoretisch analysiert als auch praktisch getestet. Dazu wird unter anderem untersucht, wie die Dokumentation gestaltet ist und welche Datenformate und Schnittstellen unterstützt werden. Im Anschluss werden verschiedene Szenarien aus den Bereichen Filtern, Analyse, Ergänzen von Inhalten und Anreicherung von Daten getestet. Die Arbeit zeigt, dass beide Programme unterschiedliche Stärken und Schwächen haben. Catmandus Stärke ist ein leichterer Einstieg in das Programm und vielfältige Optionen, bibliothekarische Datenformate und Schnittstellen zu nutzen. Ein Vorteil von KNIME ist, dass nach einer gewissen Einarbeitung viele Probleme schnell gelöst werden können und für zahlreiche Fälle spezielle Funktionen zur Verfügung gestellt werden.

Schlagwörter: Metadatenmanagement, Bibliotheken, KNIME, Catmandu, MARCXML, Software, Metadatenverwaltung

Abstract

This thesis deals with metadata management in libraries. It examines to what extent the tools KNIME and Catmandu can be used to support libraries in typical tasks of metadata management. The technical developments in the field of metadata have become more complex due to the multitude of formats, interfaces, and applications. In order to prepare and use metadata, information about the suitability of different programs is needed. KNIME and Catmandu are both theoretically analyzed and practically tested. For this purpose it is examined, among other things, how the documentation is designed, and which data formats and interfaces are supported. Typical tasks like filtering, analysis, content enhancement, and data enrichment will be tested. The work shows that both tools have different strengths and weaknesses. Catmandu's strength is an easier introduction into the program and a variety of options for using library data formats and interfaces. An advantage of KNIME is that after an initial familiarization many problems can be solved quickly and special features are made available for numerous cases.

Keywords: metadata management, libraries, KNIME, Catmandu, MARCXML, software

Inhaltsverzeichnis

Abkürzungsverzeichnis

BSZ	Bibliotheksservice-Zentrum Baden-Württemberg
DCMI	Dublin Core Metadata Initiative
DNB	Deutsche Nationalbibliothek
FAQ	Frequently Asked Questions
GND	Gemeinsame Normdatei
GPL	GNU General Public License
IMD-Typen	Inhalts-, Datenträger- und Medientyp
LoC	Library of Congress
MARC	Machine-Readable Cataloging
OAI-PMH	Open Archives Initiative Protocol for Metadata Harvesting
RDA	Resource Description and Access
RDF	Resource Description Framework
REST	Representational State Transfer

1 Einleitung

> *„Zudem stehen immer mehr Tools für die Metadatenverwaltung und -analyse zur Verfügung, welche auch von Nicht-Informatikern angewendet werden und somit neue Möglichkeiten für die Metadatenbearbeitung durch Bibliothekare und Bibliothekarinnen schaffen können.“ (Pfister, Wittwer, & Wolff, 2017)*

Zwei solcher Tools, KNIME und Catmandu, sollen in dieser Arbeit untersucht werden. Ziel ist es herauszufinden, inwiefern die beiden Programme Bibliotheken bei Aufgaben des Metadatenmanagements unterstützen können, insbesondere bei der Anwendung durch Nicht-Informatiker.

Catmandu ist ein Kommandozeilen-Tool speziell für den Umgang mit bibliothekarischen Metadaten und ist daher auf die besonderen Aufgaben und Anforderungen im Umgang mit den verschiedenen (bibliothekarischen) Datenformaten ausgelegt („LibreCat & Catmandu“, o. J.). KNIME hingegen ist ein grafisches Tool mit dem sich vielfältige Aufgaben im Bereich Data Science lösen lassen („KNIME Analytics Platform“, o. J.); es ist nicht auf den Bibliothekssektor zugeschnitten. Gerade weil die beiden Programme sich in den zentralen Elementen Zielgruppe und Benutzeroberfläche so stark unterscheiden, ist es interessant zu untersuchen, wie sich verschiedene Aufgaben des Metadatenmanagements mit ihnen lösen lassen. Dabei kann betrachtet werden, ob die Benutzeroberfläche eine Hürde ist, welche Stärken ein Tool mitbringt, das für den Umgang mit beispielsweise MARC-Daten ausgelegt ist, und welche Stärken ein Tool hat, das für ein breiteres Feld an Aufgaben gedacht ist. Ein weiterer Grund für die Wahl war, dass beide Programme Open Source sind („KNIME Analytics Platform“, o. J.; „LibreCat & Catmandu“, o. J.) und daher auch in der oft finanziell angespannten Lage von Bibliotheken oder in der Lehre (Pfeffer, 2016, S. 8) nutzbar sind.

Die Bachelorarbeit ist im Themenbereich „Metadaten“ angesiedelt, beziehungsweise in dessen Unterbereich „Metadatenmanagement“. Der Bereich Metadaten ist mit wichtigen Entwicklungen im Bibliotheksbereich wie Linked (Open) Data und dem Semantic Web verknüpft (Hipler, Prongué, & Schneider, 2018), da dort die Metadaten in Beziehung gesetzt werden können, etwa indem sie zum Resource Description Framework (RDF) migriert werden (Harlow, 2015). Zudem gibt es eine Tendenz zur Internationalisierung von Metadaten durch das Schaffen gemeinsamer Standards wie RDA. Weitere Themen im Bereich Metadaten sind die Interoperabilität zwischen den diversen Datenformaten und Offenheit der Daten als Ziel für Bibliotheken, wodurch eine neue Existenzgrundlage für diese geschaffen werden kann (Mittelbach, 2015).

Das Thema Metadatenmanagement speziell spielt in Bibliotheken aufgrund der Entwicklungen im Bereich digitaler Bibliotheken und den dazugehörigen Plattformen in den

letzten Jahren eine immer größere Rolle (Harlow, 2015). Dies liegt mitunter daran, dass das Themenumfeld der Metadaten durch die technischen Entwicklungen wesentlich komplexer geworden ist. In Bibliotheken werden immer mehr Datenformate, Schnittstellen und Schemata gleichzeitig genutzt (Pfeffer, 2016, S. 5). Durch diese vielfältigen Anforderungen ist auch das Bedürfnis nach geeigneten Tools gewachsen, mit denen die neuen Aufgaben möglichst effizient ausgeführt werden können. Dabei ist es wichtig, dass die Aufgaben schnell erledigt werden können, aber auch, dass komplexe Aufgaben mit den Tools lösbar sind. Sowohl in der Praxis als auch in der Lehre stellt sich die Frage, welches Tool welche Stärken und Schwächen hat. Diese Arbeit soll daher zwei der Tools genauer untersuchen, dabei soll veranschaulicht werden, ob bestimmte Aufgaben des Metadatenmanagements durch eine Nicht-Informatikerin damit gelöst werden können.

Dieses spezifische Thema wird in der Fachcommunity immer wieder andiskutiert, etwa beschreiben Pfister, Wittwer und Wolff (2017, S. 24f.), dass das Netzwerk Metadatenmanagement der Bibliothek der ETH Zürich sich zukünftig auch mit Tools zur Metadatenverwaltung und -analyse beschäftigen wird. Auch zum Metadatenmanagement mit FOLIO (Hemme, 2018) und Kuali-OLE (Osters, 2015) gibt es Veröffentlichungen. An der Staats- und Universitätsbibliothek Bremen wurde ein Tool entwickelt, welches das Management der Metadatenflüsse erleichtern soll (Haake & Opitz, 2017). Auch im Kontext der Europeana ist das Metadatenmanagement ein Thema (Koch & Koch, 2017).

Die Methode, mit der die Forschungsfrage beantwortet werden soll, ist ein kriterienbasierter Vergleich von KNIME und Catmandu. Dabei wird unterschieden zwischen Untersuchungskriterien und Szenarien. Untersuchungskriterien werden dabei als die spezifischen Eigenschaften und Möglichkeiten der Software verstanden. Darunter fällt beispielsweise, mit welchen Datenformaten gearbeitet werden kann, wie gut das Programm mit großen Datenmengen umgehen kann oder der Umfang und die Qualität der Dokumentation. Unter Szenarien werden typische Arbeitsprozesse des Metadatenmanagements verstanden, die in KNIME und Catmandu getestet werden sollen. In der Arbeit werden die Untersuchungskriterien und Szenarien gesammelt und entwickelt, sowie einige davon getestet. Um passende Untersuchungskriterien und Szenarien zu identifizieren, wurde in der entsprechenden Literatur recherchiert; zudem wurden Gespräche mit dem Bibliotheksservice-Zentrum Baden-Württemberg (BSZ) geführt.

Der erste Teil der Arbeit befasst sich mit den Grundlagen des Metadatenmanagements. Es wird auf die Definition des Begriffs eingegangen sowie ein Überblick über die Literatur gegeben. Zudem werden verschiedene Datenformate und Metadatenstandards sowie zwei Schnittstellen und eine Datenbank beschrieben. Dies wird für die Arbeit mit den Untersuchungskriterien benötigt. Daran anschließend werden die Tools KNIME und Catmandu vorgestellt sowie die Installationsweise beschrieben. Danach wird in einem Kapitel die Methodik beschrieben und auf die Begrenzungen dieser eingegangen. Anschließend werden die Untersuchungskriterien entwickelt und jeweils di-

rekt bearbeitet. Darauf folgt die Beschreibung und Bearbeitung der Szenarien, die in KNIME und Catmandu getestet wurden. Die Untersuchung wird mit einem Fazit beendet, das auf die Forschungsfrage eingehen soll. Fokus ist, inwiefern die Tools Bibliotheken beim Metadatenmanagement unterstützen können. Ein Ausblick auf weitere relevante Fragestellungen, die im Zusammenhang mit der Arbeit stehen, bildet den Schluss.

2 Grundlagen zum Metadatenmanagement

2.1 Metadatenmanagement in der Literatur

2.1.1 Definitionen

Die einfachste und am häufigsten aufgefundene Definition von Metadaten ist, dass diese Daten über Daten seien (Haynes, 2018, S. 9; Mitchell, 2015, S. 3; Zeng & Qin, 2016, S. 11). Daher würden Metadaten sowohl von Daten stammen als auch selbst als Daten dienen (Liu, 2007, S. 4). Haynes (2018, S. 9) schreibt hingegen, dass diese einfache, ursprüngliche Definition, den jetzigen Gebrauch nicht angemessen widerspiegeln würde. Im jetzigen Gebrauche stünde das ‚data' in ‚metadata' für Informationen, Informationsressourcen oder allgemein Einheiten, die Informationen enthalten. Damit wären dokumentarische Materialien in verschiedenen Formaten und auf verschiedenen Medien enthalten (Haynes, 2018, S. 9).

Auch Zeng und Qin (2016, S. 11) definieren Metadaten als Informationen, die beliebige Entitäten, welche Informationen enthalten, beschreiben. Cole und Foulonneau (2007, S. 111) beschreiben Metadaten als strukturierte Daten, die den Umgang mit Informationsressourcen für Bibliothekare und Bibliotheksnutzer erleichtern. Gute Metadaten könnten die Auffindung, Identifikation, Selektion, Verwaltung und Benutzung von Informationen fördern (Cole & Foulonneau, 2007, S. 111). Auch Woodley (2004) benennt im Glossar der Dublin Core Metadata Initiative (DCMI) Metadaten als strukturierte Daten über Daten. Die Metadaten seien zum Zweck der Beschreibung, Verwaltung, aus rechtlichen oder technischen Gründen, zur Benutzung und Erhaltung mit der Informationsressource verknüpft (Woodley, 2004).

Die Begriffe Metadatenverwaltung und Metadatenmanagement werden in dieser Arbeit gleichwertig verwendet. Haynes beschreibt in einem Kapitel (2018, S. 163ff.) das „management of metadata" (Haynes, 2018, S. 163) beziehungsweise „metadata management" (Haynes, 2018, S. 163) anhand eines Lebenszyklus. Der Lebenszyklus besteht aus folgenden Elementen:

- Analyse der Anforderungen an die Metadaten
- Auswahl oder Entwicklung der Metadatenschemata
- Kodierung sowie Pflege von kontrolliertem Vokabular
- Anwendung auf die Metadaten
- Importieren der Metadaten

- Qualitätskontrolle
- Suchhilfen und Schulung der Nutzer (Haynes, 2018, S. 164)

Das Metadatenmanagement könne als Abfolge dieser Schritte verstanden werden, auch wenn ein Nutzer meist nur mit einem oder zwei der Schritte befasst wäre (Haynes, 2018, S. 163).

Die ETH-Bibliothek, an der es ein Team für das Metadatenmanagement gibt, nennt als Definition die „Aufbereitung von Metadaten, automatisierte Verfahren, [einen] Überblick über neue und bestehende Standards [und] Format- und Regelwerkskenntnisse“ (Bissegger & Wittwer, 2016, S. 3).

Westbrooks (2005) hat eine etwas umfassendere und greifbarere Definition:

> *„Metadata management is the sum of activities designed to create, preserve, describe, maintain access, and manipulate metadata, MARC and otherwise, that may be owned, aggregated, or distributed by the managing institution. These organizational and intellectual activities require the physical resources (web services, scripts and cross-walks), financial commitment (much like that already invested into OPACs), and policy planing that codifies the guiding framework within which metadata exists.“ (Westbrooks, 2005)*

Auf Basis dieser Definitionen werden in dieser Arbeit unter Metadatenmanagement alle Prozesse verstanden, die anfallen, wenn man bestehende Metadaten verwendet oder anpasst, neue Metadaten erfasst, neue Konzepte für Metadaten entwickelt oder sich mit den dafür nötigen Grundlagen befasst. Eingeschränkt wird es auf Metadaten, die im informationswissenschaftlichen Kontext anfallen, also Metadaten über Einheiten, die selbst Informationen enthalten.

Der Begriff Metadatenverarbeitung, der auch in manchen Quellen verwendet wird, beschreibt in dieser Arbeit nur ein Teilgebiet des Metadatenmanagements. Dieses Teilgebiet umfasst die Anpassung der Metadaten, um sie für die eigenen Zwecke verwenden zu können – Fokus ist also die Manipulation der Daten. So schreibt Haynes (2018, S. 169), dass bei der Verwendung von bereits bestehenden Metadaten diese eventuell gesäubert und auf des Zielsystem angepasst werden müssen.

2.1.2 Literaturüberblick

Arbeiten, die ein ähnliches Thema wie diese bearbeitet haben, sind die Masterarbeit von Ursula Klute (2018) mit dem Titel „ETL-Prozesse für bibliothekarische Metadaten: Die Migration lokaler Katalogisate im GBV“ und ein Artikel von Christina Harlow (2015) in Code4Lib mit dem Titel „Data Munging Tools in Preparation for RDF: Catmandu and LODRefine“. Klute befasste sich in ihrer Arbeit mit der Migration von lokalen Katalogisaten in die zentrale Verbunddatenbank (Klute, 2018, S. III). Dabei werden auch ver-

schiedene Tools für das Datenmanagement vorgestellt, unter anderem Catmandu, OpenRefine und Mable+ sowie MARCel (Klute, 2018, S. 37ff.). Harlow beschreibt, wie an der Bibliothek der University of Tennessee Knoxville (UTK) Catmandu und LODRefine verwendet werden, um bestehende Daten in RDF umzuwandeln und wie die Daten dabei bearbeitet werden (Harlow, 2015).

Vermisst wird in der Literatur eine allgemeine Einführung in das Thema Metadaten*management*. Während es viel Literatur über die verschiedenen Formate und Schemata gibt und auch das Thema Qualitätssicherung immer wieder aufgegriffen wird (Chen, Wen, Chen, Lin, & Sum, 2011; Mitchell, 2015, S. 188ff.; Zeng & Qin, 2016, S. 317ff.), fehlt eine Einführung, in welcher der Begriff Metadatenmanagement klar und allgemeingültig definiert wird und ein Überblick über die verschiedenen Aufgabengebiete mit konkreten Beispielen gegeben wird. Stattdessen wird der Begriff meist ohne formelle Einführung verwendet, insbesondere bei Artikeln aus der Praxis. Wie im vorangegangenen Abschnitt beschrieben gibt es in Haynes' (2018, S. 163ff.) Buch ein Kapitel, in dem Metadatenmanagement anhand eines Lebenszyklus beschrieben wird. Zudem existiert eine etwas umfassendere Definition von Westbrooks (2005). Es bleibt offen, inwiefern diese Definitionen anerkannt sind und die verschiedenen Prozesse in Bibliotheken beschreiben.

Es gibt einzelne Artikel aus der Praxis, die Beispiele für die Aufgabenbereiche geben, aber meist nicht konkret werden, welche Aufgaben genau anfallen und womit gearbeitet wird. Einige Berichte stammen vom Team Metadatenmanagement der ETH-Bibliothek, etwa von einem Vortrag (Bissegger & Wittwer, 2016), in der Zeitschrift Arbido (Cavegn-Pfister, Wirth, Wittwer, & Wolff, 2017) und in b.i.t.-online (Pfister et al., 2017). Das Thema scheint allgegenwärtig zu sein, aber in der Literatur wenig auf theoretische Weise aufbereitet zu werden. Auch ein Überblick über die verschiedenen Programme, mit denen Metadaten verarbeitet werden können, wird vermisst. In einigen Quellen werden verschiedene Programme kurz angerissen, aber eine ausführliche Auseinandersetzung mit den Unterschieden und Möglichkeiten fehlt. Zu nennen für einen möglichen Überblick sind die bereits genannte Arbeit von Klute (2018), ein Abschnitt in einem Buch von Yang und Li (2015, S. 51ff.) sowie Vortragsfolien von Pfeffer (2016).

Hervorzuheben ist noch das Buch „Coding with XML for efficiencies in cataloging and metadata: practical applications of XSD, XSLT, and Xquery" von Cole, Han und Schwartz (2018). Es bietet eine Einführung in die Arbeit mit bibliografischen Daten im XML-Format. Dabei werden konkrete Beispiele gegeben und es wird in die praktische Arbeit mit MARCXML eingeführt. Ein weiteres Buch von Cole und Foulonneau (2007, S. 137ff.) ist erwähnenswert, da dort in einem Kapitel beschrieben wird, welche Anpassungen an Metadaten nach dem Harvesten über OAI-PMH nötig sein können. Des Weiteren ist als umfassende Einführung in Metadaten im Bereich Informationswissenschaften das Werk von Zeng und Qin (2016) zu nennen.

2.2 Datenformate und Metadatenstandards

2.2.1 MARC 21

MARC steht für Machine-Readable Cataloging. Mit machine-readable ist gemeint, dass ein Computer die Daten, die in einem MARC-Record enthalten sind, lesen und interpretieren kann. MARC-Records geben jene bibliografischen Daten an, die früher auf Katalogkarten zu finden waren, wie etwa eine Beschreibung der Ressource, Schlagwörter und eine Signatur (Furrie, 2009). Das MARC-Format entstand Ende der sechziger Jahre an der Library of Congress (LoC) und wurde 1968 zum ersten Mal veröffentlicht (Mitchell, 2015, S. 155). An der Library of Congress wurde es von Henriette D. Avram, einer Programmiererin, entwickelt. Laut Mitchell (2015, S. 155) war das Ziel auch, dass Katalogdaten zukünftig von Computern verarbeitet werden können.

Nach der Veröffentlichung von MARC entstand eine Reihe verschiedener MARC-Standards, die nationale oder andere spezielle Gegebenheiten berücksichtigten (Haynes, 2018, S. 56). In den neunziger Jahren wurden diese verschiedenen Standards vereinheitlicht und als MARC 21 publiziert. Der MARC 21-Standard wird von der Library of Congress gepflegt (Haynes, 2018, S. 56). Der Inhalts der MARC-Felder wird nicht durch MARC geregelt, sondern durch andere Standards oder Regelwerke (Liu, 2007, S. 19), inzwischen durch das Katalogisierungsregelwerk Resource Description and Access (RDA) („RDA in MARC“, 2017).

MARC 21 wird als Beispiel für die ISO Norm 2709 („Information and documentation — Format for information exchange“) angesehen (Zeng & Qin, 2016, S. 25). Dabei wird MARC von Zeng und Qin (2016, S. 23) als ein Standard für die Datenstruktur und ISO 2709 (MARC) als ein Standard für den Datenaustausch beschrieben. Standards für den Datenaustausch werden auch als Formate bezeichnet (Zeng & Qin, 2016, S. 25).

Das MARC-Vokabular beschreibt die verschiedenen Elemente des Standards: „field, tag, indicator, subfield, subfield code, and content designator“ (Furrie, 2009). Ein MARC-Record besteht aus laut Furrie (2009) aus verschiedenen *fields*, Feldern. So gibt es beispielsweise Felder für den Titel und den Autor. Jedem Feld ist ein *Tag* zugeordnet. Dieser besteht aus drei Ziffern. Der Tag gibt also an, um was für ein Feld und damit um was für eine Information es sich handelt (Furrie, 2009). Hinter dem Tag folgen die zwei *indicators*, Indikatoren. Diese bestehen jeweils aus einer Zahl zwischen 0 und 9. Es gibt Felder, bei denen beide Indikatoren belegt sind, oder keine, oder einer von beiden. Nicht belegte Indikatoren sollten durch die Raute # markiert werden (Furrie, 2009). Felder können unterteilt sein in *subfields*, Unterfelder. Diese werden durch einen *subfield code*, kurz Code, markiert. Der Code besteht meist aus einem Trennzeichen (oft dargestellt durch das Dollarzeichen $) und einem kleingeschriebenen Buchstaben, manchmal auch einer Zahl. Wiederum gibt der Code an, um was für eine Art von Information es sich handelt. Der Begriff *Content Designator* bezeichnet die Ge-

samtheit der Tags, Indikatoren und Codes (Furrie, 2009). Zu einem MARC-Record gehört außerdem noch der *Leader*, die 24 ersten Zeichen, die hauptsächlich von Computern verarbeitet werden. Im Kommunikationsformat beginnt ein MARC-Record mit dem Leader. Darauf folgt ein *directory*. Dies gibt die im jeweiligen Record verwendeten Tags an sowie die Position, an der das jeweilige Feld beginnt. Daran schließen sich die Informationen aus den Feldern an (Furrie, 2009).

Die Library of Congress publiziert online das *MARC 21 Format for Bibliographic Data* sowohl in der ausführlichen („full“) und der kurzen („concise“) Version. Dort wird, gruppiert nach den Tags, jedes Feld erklärt und Beispiele für die Benutzung gegeben (Network Development and MARC Standards Office, Library of Congress, 2019).

2.2.2 MARCXML

Unter MARCXML kann man MARC im XML-Format statt im binären Format verstehen – dadurch sei es einfacher zu verarbeiten und zu teilen (Cole et al., 2018, S. 21). Es gibt verschiedene Möglichkeiten, MARC 21 in XML auszudrücken (zu serialisieren). Das 2002 vorgestellte MARCXML sei jedoch der De-facto-Standard, um MARC in XML zu serialisieren (Cole et al., 2018, S. 22). In MARCXML gibt es nur wenige definierte Elemente: zwei Elemente auf oberster Stufe „(<collection>,<record>) [und die Elemente] <leader>, <controlfield>, <datafield>, and <subfield>“ (Cole et al., 2018, S. 23). Laut Mitchell (2015, S. 160) kann ein MARCXML-Dokument entweder einen oder mehrere Records beschreiben. Die jeweiligen Tags und Codes der (Unter)Felder werden durch die Attribute des jeweiligen XML-Elements ausgedrückt (Cole et al., 2018, S. 23). Das Feld 245 könnte etwa wie folgt aussehen:

```
    <datafield tag="245" ind1="1" ind2="0">
       <subfield code="a">Manche lachen keiner weint</subfield>
       <subfield code="c">Hansgert Lambers</subfield>
    </datafield>
```

(aus: Atest.mrc.xml vom 25.06.2019, https://data.dnb.de/testdat/)

Ein großer Vorteil dieser Herangehensweise ist, dass dadurch alle Informationen, die im binären MARC-Format vorhanden sind, auch in MARCXML abgebildet werden können – die Umwandlung ist verlustfrei (Cole et al., 2018, S. 23; Zeng & Qin, 2016, S. 407). Außerdem können dadurch leicht zusätzliche Felder im 9XX-Bereich definiert werden, um das Format an lokale Gegebenheiten anzupassen (Cole et al., 2018, S. 24).

Wichtige Begrifflichkeiten im Umgang mit XML sind *Elemente, Attribute, Start- und End-Tags, Wurzelelemente* und *Unterelemente*. Unter Elementen versteht man die logische Struktureinheit eines XML-Dokuments. Die Elemente setzen sich zusammen aus einem Start-Tag und einem End-Tag, zwischen denen sich der Inhalt befindet (Bray, Paoli, Sperberg-McQueen, Maler, & Yergeau, 2013a). Im obigen Beispiel ist </datafield> etwa ein End-Tag. Attribute dienen der Spezifizierung (Bray et al., 2013b)

und sind im Beispiel etwa tag="245". Als Wurzelelement wird das alles umschließende Element auf der obersten Hierarchiestufe beschrieben, Unterelemente sind alle Elemente, die sich in diesem Wurzelelement befinden. Jedes XML-Dokument muss ein Wurzelelement haben (Bray, Paoli, Sperberg-McQueen, Maler, & Yergeau, 2013b).

Gepflegt wird der MARCXML-Standard von der Library of Congress beziehungsweise deren Network Development and MARC Standards Office (Cole & Foulonneau, 2007, S. 122; Zeng & Qin, 2016, S. 407). Vorteile von MARCXML gegenüber MARC sind, dass es leichter gelesen und verbessert werden kann (Cole et al., 2018, S. 22). Die Daten können leichter umgewandelt oder ausgetauscht werden, insbesondere mit anderen XML-Daten (Zeng & Qin, 2016, S. 407). Das bedeutet auch, dass es für Programmierer leichter ist, auf einzelne Bestandteile eines MARC-Records zuzugreifen (Mitchell, 2015, S. 158).

2.2.3 RDF

Das Resource Description Framework (RDF) an sich ist im Gegensatz zu etwa XML kein spezifischer Kodierungsstandard, sondern eine logische Struktur (Liu, 2007, S. 96). Zeng und Qin (2016, S. 67) beschreiben RDF als Framework, mithilfe dessen Informationen im Internet abgebildet werden können. RDF kann in verschiedenen Formaten serialisiert werden. Ursprünglich war es auf XML-basiert, inzwischen gibt es viele verschiedene Möglichkeiten. Einige davon sind etwa RDF/XML, RDFa, N-triples, Turtle und JSON-LD. Daher solle RDF als Datenmodell angesehen werden und nicht als spezifische Serialisierung (Zeng & Qin, 2016, S. 133).

RDF gilt Graphdatenbank, deren Ziel das Treffen von Aussagen über und die Herstellung von Beziehungen zwischen Ressourcen ist (Mitchell, 2015, S. 132). Eine einzelne Einheit in RDF ist ein *triple*. Dieses besteht aus drei Einheiten, einem Subjekt, einem Prädikat und einem Objekt. Das Subjekt wird mithilfe des Objekts beschrieben. Das Prädikat beschreibt dabei die Art der Relation (Mitchell, 2015, S. 132). Daher bietet jedes dieser Triple auch ohne die anderen Triple in einem Graph eine Information. Demgegenüber lässt ein einzelnes Element eines XML-Dokuments ohne seinen Kontext keine Aussagen über die Relation zu (Mitchell, 2015, S. 136).

Ein Nutzen von RDF ist etwa, dass mit RDFa beschriebene Webseiten es den Suchmaschinen-Crawlern sehr viel leichter machen, die Webseiten zu indexieren. Zudem können die Ersteller der Seiten festlegen, auf welche Daten sich der Crawler konzentrieren soll (Mitchell, 2015, S. 136). Dies zeigt auf, inwiefern es durch RDF möglich wird, Beziehungen zwischen Informationen maschinenlesbar aufzubereiten. Im Bibliotheksbereich bietet etwa die Deutsche Nationalbibliothek (DNB) ihre Daten im Rahmen des Linked-Data-Services als RDF an („Linked-Data-Service", o. J.). Dadurch soll die Nachnutzung der Daten erleichtert werden. Verfügbar sind die Titeldaten als Turtle und als RDF/XML. Die Normdaten der Gemeinsamen Normdatei (GND) werden als Turtle

und als JSON-LD bereitgestellt. Für die Umwandlung der Daten wird die Software Metafacture verwendet („Linked-Data-Service“, o. J.).

2.2.4 Dublin Core

Dublin Core ist ein Metadatenstandard, der vom Usage Board der Dublin Core Metadata Initiative („DCMI Usage Board“) gepflegt wird. Dieses überarbeitet die DCMI Metadata Terms und ist zuständig für die ISO-Norm 15836 („DCMI: DCMI Usage Board“, 2019). Es gibt „einfaches“ („simple“) Dublin Core mit 15 Kernelementen und „qualified“ Dublin Core, das weitere Elemente und „qualifying terms (additional elements in XML)“ (Cole et al., 2018, S. 26) enthält. Die 15 Kernelemente des einfachen Dublin Core sind Bestandteil der ISO-Norm 15836 („ISO 15836-1“, 2017).

Die Dublin Core Metadata Initiative stellt verschiedene Spezifikationen zur Verfügung. Den Status „Recommendation“ haben vier Spezifikationen, die sich mit der Auszeichnung von Dublin Core beschäftigen. Diese umfassen Hinweise zur Verwendung von Dublin Core als HTML/XHTML, RDF, RDF/XML und XML („DCMI: Recommendation“, 2019).

Dublin Core kann zur Beschreibung digitaler Ressourcen in vielen verschiedenen Formaten, zum Beispiel für Bilder, Text, audiovisuelle Inhalte oder Karten verwendet werden. Es ist der Metadatenstandard bei DSpace (Software für Repositorien) und eine Voraussetzung für die Nutzung von OAI-PMH (Cole et al., 2018, S. 26). Cole et al. (2018, S. 27) schreiben, dass die Schlichtheit und der allgemein gehaltene Aufbau der Dublin Core-Elemente dazu geführt haben, dass es nun so bekannt und weit verbreitet ist.

2.3 Schnittstellen und Datenbanken

2.3.1 OAI-PMH

OAI-PMH steht für Open Archives Initiative Protocol for Metadata Harvesting. Es handelt sich dabei um ein eng definiertes Protokoll, das für Computer-zu-Computer-Interaktionen gedacht ist. Es wurde im Januar 2001 veröffentlicht (Cole & Foulonneau, 2007, S. xiii). Es entstand aus der Eprint-Community, denn dort wurde ein Protokoll benötigt, mit dem die verschiedenen Repositorien auf einfache Weise geladen und genutzt werden können (Mitchell, 2015, S. 177). Mit dem Protokoll können beschreibende Metadaten geteilt werden (Cole & Foulonneau, 2007, S. 3). Unter beschreibenden Metadaten werden dabei solche verstanden, die nützlich für Discovery, Auffindung, Klassifizierung, Gruppierung, in Beziehung setzen, Interpretation und Identifizierung der Ressource sind (Cole & Foulonneau, 2007, S. 4).

Die Daten werden dabei via HTTP übertragen, verwendet werden HTTP GET und HTTP POST (Mitchell, 2015, S. 177).OAI-PMH ist im Vergleich zu anderen Schnittstellen einfach und technisch simpel gehalten (Cole & Foulonneau, 2007, S. 3). Als Datenformat kommt XML zum Einsatz (Mitchell, 2015, S. 177). Es ist Pflicht, dass Metadaten im Dublin Core-Format zur Verfügung gestellt werden, damit eine gewisse Interoperabilität besteht. OAI-PMH kann allerdings für alle Metadatenstandards verwendet werden, die als XML serialisiert werden können. Daher bieten Bibliotheken oft zusätzlich zu Dublin Core auch MARCXML-Daten an (Mitchell, 2015, S. 178).

OAI-PMH ist nicht dafür ausgelegt, dynamische Echtzeit-Suchen durchzuführen (Cole & Foulonneau, 2007, S. 15). Stattdessen werden die Metadaten eingesammelt und an einem Ort abgelegt. Sie verbleiben also nicht bei den Erstellern, an die dann Anfragen gesendet werden, wie das bei einigen anderen Schnittstellen der Fall ist (Cole & Foulonneau, 2007, S. 16).

Im Bibliotheksbereich sowie in Archiven und Museen wird OAI-PMH häufig genutzt, um Metadaten aufzufinden, zu verteilen und zu aggregieren (Mitchell, 2015, S. 177). Auch in digitalen Bibliotheken wie der Digital Public Library of America kommt OAI-PMH zum Einsatz (Mitchell, 2015, S. 184). Ebenso verwenden viele Repositorien OAI-PMH („Library Carpentry“, o. J.). Beispielsweise ist es auch in der Repositorien-Software Fedora integriert („Features - Fedora“, o. J.). Auch das Open-Source-Bibliotheksmanagementsystem Koha unterstützt OAI-PMH zumindest zum Teil („APIs and protocols supported by Koha“, 2019).

2.3.2 REST

Der REpresentational State Transfer (REST) ist ein Standard für Computer-zu-Computer-Interaktionen im Web („What is REST?“, o. J.). Systeme, die sich an dem Standard orientieren, werden auch RESTful genannt. Charakteristika von REST sind die Trennung von Server und Client und die Zustandslosigkeit. Die Trennung von Client und Server bedeutet, dass beide Elemente unabhängig voneinander verändert werden können, ohne dass es Einfluss auf die jeweils andere Seite hat. Mit der Zustandslosigkeit ist gemeint, dass weder Server noch Client wissen müssen, in welchem Zustand sich der jeweils andere Part befindet („What is REST?“, o. J.). Der Standard wurde 2000 von Roy Fielding vorgeschlagen („What is REST – Learn to create timeless REST APIs“, o. J.). REST verwendet HTTP und zwar GET, POST, PUT und DELETE („What is REST?“, o. J.).

Genau wie OAI-PMH wird REST von vielen Anwendungen im Bibliotheksbereich unterstützt, etwa von Repositorien („Library Carpentry“, o. J.), der Software Fedora („Features - Fedora“, o. J.), dem Bibliotheksmanagementsystem Koha („APIs and protocols supported by Koha“, 2019) und der Digital Public Library of America (Karadkar, Altman, Breedlove, & Matienzo, 2016).

2.3.3 MongoDB

Bei MongoDB handelt es sich um eine dokumentenorientierte Datenbank („What Is MongoDB?“, o. J.). Die Daten werden dabei JSON-ähnlichen Dokumenten gespeichert, wodurch die Felder sich von Dokument zu Dokument unterscheiden können und die Datenstruktur mit der Zeit verändert werden kann („What Is MongoDB?“, o. J.). Ein Eintrag ist in MongoDB je ein Dokument, dessen Datenstruktur sich aus Feld-Wert-Paaren zusammensetzt. Die Felder können andere Dokumente, Arrays oder Arrays von Dokumenten enthalten („Introduction to MongoDB“, o. J.). Dokumente können dabei in „collections“ („Introduction to MongoDB“, o. J.) gespeichert werden, dies entspricht den Tabellen in relationalen Datenbanken. MongoDB, Inc. stellt die Datenbank her. Es gibt eine kostenlose MongoDB Server Version, aber MongoDB, Inc. bietet auch Installationen in der Cloud, Support und vieles mehr an („FAQ“, o. J.-a).

3 Einführung zu den Tools KNIME und Catmandu

3.1 KNIME

3.1.1 Vorstellung

2004 wurde mit der Arbeit an KNIME („Konstanz Information Miner“) begonnen. Die erste Version wurde 2006 veröffentlicht („KNIME Open Source Story“, o. J.). Das Team arbeitete an der Universität Konstanz und bestand damals aus Entwicklern einer Software Firma, die Tools für den Pharmaziebereich herstellte. Es war von Beginn an als Open-Source-Tool ausgelegt. Ziel war es auch, Möglichkeiten zur Kollaboration zu bieten, Forschung zu betreiben und große Mengen sehr heterogener Daten zu verarbeiten („KNIME Open Source Story“, o. J.). Nach der ersten Veröffentlichung 2006 gab es bereits einige Softwarehersteller, die Tools basierend auf KNIME erstellten. Inzwischen gibt es die KNIME AG als Muttergesellschaft. Neben der Open-Source-Basis bietet die KNIME AG auch Erweiterungen an, die kommerzielle Software nutzen und daher lizenziert werden müssen. Diese kommerziellen Erweiterungen werden als Jahreslizenz verkauft, wovon ein Teil der Einnahmen der Weiterentwicklung der Open-Souce-Basis dient. Benutzt wird KNIME laut der Webseite von großen Unternehmen aus vielen verschiedenen Branchen, etwa den Lebenswissenschaften, der Finanzwelt, von Verlagen, im Onlinehandel und mehr („KNIME Open Source Story“, o. J.).

KNIME ist unter der GNU General Public License (GPL), Version 3 lizenziert, mit bestimmten zusätzlichen Genehmigungen nach Sektion 7 der GPL. Die Webseite („KNIME Open Source Story“, o. J.) fasst zusammen, dass man KNIME ohne Einschränkung herunterladen und verwenden darf, dass es allerdings keine Garantie gibt. Ohne Modifikationen darf KNIME ohne Einschränkungen verbreitet werden. Bei Änderungen an KNIME soll man die Lizenz lesen. Wenn man neue Nodes entwickeln möchte, darf man diese unter jeder Lizenz, die man nehmen möchte, veröffentlichen – für Nodes gilt die GPL quasi nicht, da sie kein Derivat von KNIME sind (dies ist in Sektion 7 geregelt). Außerdem wird durch die Ausnahme ermöglicht, dass proprietäre Software aus KNIME heraus über die API angesteuert wird („KNIME Open Source Story“, o. J.).

KNIME ist modular aufgebaut („KNIME Open Source Story“, o. J.). Es gibt eine Basis und viele verschiedene *extensions*, Erweiterungen. Mit diesen können verschiedene Funktionalitäten dem Programm hinzugefügt werden. Erweiterungen gibt es von KNIME selbst, von der Community und von Drittanbietern („Install Extensions and Integrations“, o. J.).

Basis der Arbeit mit KNIME sind sogenannte *Nodes*. Jeder Node erfüllt eine spezifische Aufgabe, etwa das Einlesen einer Datei oder das Filtern nach bestimmten Werten. Die Nodes können miteinander verknüpft werden, sodass der Output des ersten Nodes der Input des zweiten Nodes ist. Dadurch lassen sich auch komplexe Workflows abbilden („Build a workflow“, o. J.). Die Daten werden dabei als Tabellen angezeigt, sodass neue Spalten oder Zeilen eingefügt werden können oder zwei Tabellen miteinander verknüpft werden können.

3.1.2 Installation

Sowohl KNIME als auch Catmandu wurden auf einem Rechner mit dem Betriebssystem Ubuntu 18.04 (Bionic Beaver) 64-bit mit der Standard-Desktop-Umgebung GNOME (Version 3.28.2) installiert.

Für KNIME wurde zuerst von der Download-Seite („Download KNIME Analytics Platform“, o. J.) die Version für Linux heruntergeladen. Dabei handelt es sich um ein Tar-Archiv, das dann entpackt werden muss. Danach lässt sich KNIME durch die ausführbare Datei in dem entpackten Ordner starten.

Beim ersten Testen des Programms fallen verschiedene Probleme auf. In den Beispielen können die Nodes nicht ausgewählt werden. Zudem werden die Beispiele in Tabs ohne Namen geöffnet, sodass die Tabs nicht sichtbar sind, wenn man sich in einem anderen Tab befindet. Bei dem Versuch, eines der Beispiele in einem eigenen Worspace nachzubauen, kommt beim Einlesen der CSV-Datei aus dem Beispiel eine kryptische Fehlermeldung: „Unable to parse xml: line=1: Content in not allowed in prolog. xml: URI=java.io.FileInputStream@4977e527 dtd: URI=null“. Bei der Suche nach einer Lösung taucht ein Video auf, das den File Reader-Node erklären soll. In diesem ist dann zu sehen, dass das Dialogfenster eigentlich mehrere Optionen zur Auswahl bietet. In der installierten Version ist nur ein großes graues Fenster zu sehen, bei dem oben über File – Load Settings eine Datei eingelesen werden kann.

Bei der darauf folgenden Recherche, warum die Inhalte des Dialogsfensters nicht korrekt angezeigt werden, findet sich in den Frequently Asked Questions (FAQ) ein ähnliches Problem. Das Problem dort bezieht sich auf Fedora 10 mit GNOME, Dialogfenster werden dort nicht „repainted“ („FAQ | KNIME“, o. J.). Als Antwort wird vorgeschlagen, dass libgxim-Paket zu deinstallieren. Dieses scheint unter dem genutzten System nicht installiert zu sein. Daher wird mit dem Begriff „repainted“ weiter gesucht. Die Suche „knime file reader node configure dialog not repainted“ führt schließlich zu einem Foreneintrag („Knime Config Dialogs/Popups Blank“, 2017), in dem andere dasselbe Problem haben, wenn sie per Fernzugriff von CentOS zugreifen. Später beschreibt jemand das gleiche Problem unter Fedora29 ohne Remote Zugriff und löst es durch die Nutzung des Windows Builds via Wine. Daher wird dieser Weg getestet.

Mithilfe von Wine lassen sich auch einige Windows-Programme unter Linux ausführen. Daher wurde die ZIP-Archiv-Version für Windows heruntergeladen und entpackt. Nun muss man in der Kommandozeile in den Ordner navigieren, in dem sich die Datei knime.exe befindet. Mit dem Befehl „wine start knime.exe“ kann dann KNIME in der Windows-Version gestartet werden. Die bisherigen Probleme treten nicht mehr auf. Schlussendlich installiert ist nun KNIME in der Version 4.0.0 für Windows.

3.2 Catmandu

3.2.1 Vorstellung

Catmandu wird durch die Universitäten Lund, Gent und Bielefeld im Rahmen des Open-Source-Projekts LibreCat entwickelt („About LibreCat“, o. J.). Inzwischen sind auch einige unabhängige Entwickler beteiligt. Der Anfang des Projekts war 2010, als entschieden wurde, die bisherige Software im großen Stil umzugestalten und als Open-Source-Software zur Verfügung zu stellen. Lizenz ist auch hier die GNU General Public License der Free Software Foundation, aber in der Version 2 oder höher („About LibreCat“, o. J.).

Ziel mit der Entwicklung der Software Catmandu war es, Bausteine bereitzustellen, die immer wieder genutzt werden können, wenn Repositorien oder ähnliche Anwendungen erstellt werden sollen („About LibreCat“, o. J.). Denn laut den Entwicklern werden oft verschiedene Systeme gleichzeitig erstellt und installiert, deren Aufgaben sich teilweise überschneiden, die aber zu unterschiedlichen Ergebnissen führen. Die Systeme unterscheiden sich etwa in den Nutzergruppen oder Erschließungsregeln. Daher sollten wiederverwendbare Tools erstellt werden („About LibreCat“, o. J.).

Catmandu ist ein Kommandozeilentool, mit dem auf Daten zugegriffen und diese angepasst werden können. Die Zielgruppe sind dabei digitale Bibliotheken, Forschungsdienste oder andere Open-Data-Anwendungen (Catmandu, o. J.-a). Daher werden auch spezifische bibliothekarische Formate wie MARC oder MODS unterstützt (Catmandu, o. J.-a). Catmandu wird beispielsweise in der Zeitschriftendatenbank (ZDB) verwendet, um die Datensätze zu bearbeiten („Use Cases“, o. J.). Die entwickelnden Bibliotheken nutzen Catmandu vorrangig, um ihre Daten in den Datenbanken aufzubereiten, damit sie in Repositorien genutzt werden können („Use Cases“, o. J.).

Catmandu ist modular aufgebaut, wie auch KNIME. Ziel war es, ein Basis-Programm mit Plug-ins zu erstellen („About LibreCat“, o. J.). In Catmandu werden diese Plug-ins als Module bezeichnet (Catmandu, o. J.-b). Es gibt eine Liste mit *distributions* („Distributions“, o. J.). Jede Distribution enthält verschiedene Module und dient der Installation via CPAN. Im CPAN (Comprehensive Perl Archive Network) wird Perl-Code und die zugehörige Dokumentation gesammelt und zur Verfügung gestellt („MetaCPAN About“, o. J.).

Weitere wichtige Begriffe im Kontext von Catmandu sind Importer, Exporter, Stores und Fixes (Catmandu, o. J.-c). Mit Importern können Daten eingelesen werden, etwa im MARC- oder CSV-Format oder auch über Schnittstellen wie OAI-PMH. Mit Exportern können die Daten wieder in diese Formate umgewandelt werden. Als Stores werden Datenbanken bezeichnet. Mit Fixes können die Daten verändert werden (Catmandu, o. J.-c). Fix ist die Domain Specific Language in Catmandu, die es für Nicht-Programmierer leichter machen soll, die Daten zu manipulieren (Catmandu, o. J.-d).

Ein Catmandu-Befehl in der Kommandozeile kann wie folgt lauten:

```
catmandu convert MARC to CSV --fix 'marc_map(245,title); retain(tit-
le)' < data.mrc
```

(„Catmandu::MARC::Tutorial“, o. J.)

Mit „catmandu“ wird signalisiert, dass das Programm Catmandu verwendet werden soll. „convert“ gibt an, dass die Daten eingelesen und umgewandelt werden sollen. Mit „--fix“ wird angegeben, dass nun die entsprechenden Fixes folgen und mit „< data.mrc“ wird festgelegt, dass der Input aus der Datei data.mrc stammt. Zusätzlich könnte noch die Angabe „> data2.mrc“ vorhanden sein. Dann würde der Output in die Datei data2.mrc gespeichert werden. Alternativ ist die Angabe „| less“ hilfreich, um sich den Output des Programms Seite für Seite anzusehen („Bash-Skripting-Guide für Anfänger › Shell“, 2019). Zudem wird ein Texteditor benötigt, um Dateien mit mehreren Fixes zu schreiben. In dieser Arbeit wurde dafür nano verwendet. Mit „nano my_fixes.txt“ wird beispielsweise eine Datei mit dem Namen my_fixes.txt angelegt und aufgerufen. Zu beachten ist, dass immer der ganze Pfad zur Datei angegeben werden muss, wenn man sich nicht im entsprechenden Ordner befindet. Da Daten im YAML-Format für das menschliche Auge leicht lesbar sind, wird in dieser Arbeit oft in dieses Format umgewandelt. Es sind aber auch alle anderen Formate nutzbar.

3.2.2 Installation

Es gibt viele Wege, um Catmandu zu installieren. Allein für Debian-basierte Systeme wie Ubuntu gibt es verschiedene Möglichkeiten (Catmandu, o. J.-b) mit verschiedenen Vor- und Nachteilen:

- cpanm – Vorteil: alle Distributionen verfügbar, Nachteil: unklar, wie Distributionen wieder entfernt werden können
- apt – Vorteil: üblicher Paketmanager in Ubuntu, Nachteil: nicht alle Distributionen verfügbar
- den Quellcode kompilieren – sollte eigentlich nicht nötig sind
- den Quellcode zu kompilieren und dabei möglichst viele Pakete aus den offiziellen Quellen zu nehmen – Nachteil: komplex

- Docker – Nachteil: kein Vorwissen in der Verwendung vorhanden, unklar, wie Distributionen nachinstalliert werden können

Zudem gibt es einen Download auf der Webseite („LibreCat & Catmandu“, o. J.), was unter Linux aber eigentlich kein üblicher Installationsweg ist. In der Dokumentation sind lediglich die verschiedenen Wege genannt und beschrieben. Es gibt jedoch keine Angaben, was zu bevorzugen ist und auch auf die Unterschiede wird nicht eingegangen. Da via CPAN alle Distributionen verfügbar sind und bei Nutzung von cpanminus (cpanm), welches auch von Catmandu empfohlen wird, eine Deinstallation möglich sein soll, wurde dieser Installationsweg gewählt. Dabei wurde der Anleitung in der Dokumentation (Catmandu, o. J.-b) gefolgt und auf die Empfehlungen zu den wichtigsten Modulen zurückgegriffen. Somit wurden neben dem Basismodul Catmandu auch die Module zu MARC, OAI, RDF und XLS installiert. Installiert ist Version 1.2002. Im Nachhinein hätte für die in der Arbeit untersuchten Szenarien vermutlich auch eine Installation mit apt gereicht, da die nötigen Distributionen verfügbar gewesen wären, dies wurde jedoch nicht getestet.

4 Methodik

Ziel der vorliegenden Arbeit ist die Analyse der Programme KNIME und Catmandu hinsichtlich ihrer Möglichkeiten, für Bibliotheken im Rahmen des Metadatenmanagements von Nutzen zu sein. Dazu sollen sie sowohl hinsichtlich ihrer Funktionalität im Allgemeinen und ihrer Bedienung im Speziellen geprüft werden. Dazu wurde eine Unterscheidung in Untersuchungskriterien und Szenarien vorgenommen.

Unter Untersuchungskriterien werden objektiv bewertbare Kriterien verstanden, die sich vorrangig mit den Rahmenbedingungen und den Funktionen des Programms befassen. Diese Funktionen werden nicht praktisch erprobt, sondern nur hinsichtlich des Vorhandenseins überprüft. Hintergrund ist, dass aufgrund des Umfangs der Arbeit nicht alle Funktionen, wie etwa die Nutzung verschiedener Schnittstellen, praktisch getestet werden können. Zudem sollen Faktoren, die die Nutzbarkeit einer Software maßgeblich beeinflussen, möglichst faktenbasiert untersucht werden. Die Szenarien hingegen sind typische Vorgänge im Metadatenmanagement in Bibliotheken, mithilfe derer die Software praktisch getestet werden soll. Die Szenarien sind wichtig, um darstellen zu können, wie an ein Problem herangegangen werden kann, welche Probleme bei der Bearbeitung auftauchen und wie diese gelöst werden können.

Aufgrund des Rahmens der Arbeit handelt es sich bei den Ergebnissen dieser Untersuchungen um – zu einem gewissen Grad – subjektive Empfindungen. Die Szenarien und Kriterien werden jeweils nur von einer Person untersucht, sodass deren Vorkenntnisse und vorherige Erfahrungen einen Einfluss auf das Ergebnis haben werden. Daraus kann nicht zwangsläufig geschlossen werden, wie es anderen Personen, deren Vorkenntnisse andere sind, mit den Aufgabestellungen umgehen würden. Nichtsdestotrotz soll mit dieser Arbeit ein Einblick gegeben werden, wie jemand mit einem bibliothekarischen Hintergrund, aber ohne Informatik-Erfahrung, die entsprechenden Programme benutzen kann. Zum Vorwissen sei gesagt, dass aufgrund der vorherigen Verwendung von Ubuntu als Betriebssystem in begrenztem Umfang Vorerfahrungen mit der Kommandozeile und dem Umgang mit unerwarteten Problemen bei der Verwendung von Software vorhanden waren. Neben der Subjektivität durch das Vorhandensein bzw. Fehlen von Vorerfahrungen könnte auch die technische Basis einen Einfluss auf die Lösung mancher Probleme haben.

Die konkrete Beschreibung des jeweiligen Kriteriums beziehungsweise Szenarios erfolgt direkt vor der Bearbeitung beziehungsweise dem Ergebnis der Untersuchung. Damit soll gewährleistet werden, dass die genauen Rahmenbedingungen noch präsent sind.

Zu Beginn der Arbeit wurden alle Faktoren, die für die Bewertung der Software eine Rolle spielen könnten, gesammelt. Diese Faktoren stammten aus der Literatur sowie

aus Gesprächen und eigenen Überlegungen. Anschließend wurden die Faktoren sortiert und schließlich in die Kategorien „Untersuchungskriterien“ und „Szenarien“ unterteilt.

Nach Möglichkeit wurde auf Daten zurückgegriffen, die frei verfügbar waren, damit die Beispiele gegebenenfalls nachvollzogen werden können. Da die Daten jedoch größtenteils vom Server mit Testdaten der DNB („Index of /testdat“, o. J.) stammten, wurden die Datensätze inzwischen teilweise bereits wieder durch neue Daten ersetzt.

Für die Untersuchungskriterien wurden zuerst alle Ideen aus Literatur, Gesprächen und eigenen Überlegungen gesammelt. Diese wurden daraufhin sortiert und es wurden Kategorien gebildet. Diese sind: Plattformen und Aktualität, Support und Dokumentation, Datenformate und Metadatenstandards, Schnittstellen und Datenbanken sowie Leistungsfähigkeit. Im Anschluss wurden zu den jeweiligen Kategorien objektiv untersuchbare Fakten gesammelt. Diese wurden beschrieben und im Anschluss notiert, ob sie im Rahmen der Arbeit untersucht werden können. An diese jeweilige Erläuterung der Kategorie schließt sich die Untersuchung des Kriteriums in KNIME und in Catmandu an. Dabei wurde zu den jeweiligen Kriterien recherchiert, die Ergebnisse der Recherchen genannt und begründet, was dies für die Einschätzung des Programms bedeutet.

Die Szenarien wurden ebenfalls in mehreren Schritten entwickelt. Hier wurde ebenfalls in der Literatur recherchiert. Da diese jedoch oft etwas allgemeiner gehalten ist, wurde zusätzlich noch Kontakt zum BSZ aufgenommen. Dadurch konnten konkrete Beispiele aus dem Umgang mit E-Book-Daten und der Einspielung von Fremddaten gewonnen werden. Wieder wurden alle Szenarien gesammelt, da es jedoch viele verschiedene Arten von Szenarien gibt, mussten diese eingegrenzt werden. Für einige Szenarien wären auch weiterführende technische Kenntnisse und Möglichkeiten nötig, etwa bei der Nutzung von Datenbanken. Die Szenarien wurden schließlich auf vier Gruppen eingegrenzt: Filtern, Analyse, Ergänzen von Inhalten und Anreicherung eines Datensatzes mit Informationen aus einem anderen Datensatz. Filtern ist eine wichtige Grundvoraussetzung im Umgang mit Daten. Unter den Bereich Analyse fallen Aufgaben, die besonders dann nötig sind, wenn Fremddaten eingespielt werden sollen. Bei dieser Analyse kann beispielsweise festgestellt werden, dass ein Feld fehlt, dieses sollte dann ergänzt werden. Die Anreicherung ist wichtig, wenn mit mehr als einem Datensatz gleichzeitig gearbeitet werden soll.

Insgesamt bauen die Szenarien in gewissem Rahmen damit aufeinander auf: beim Filtern müssen einzelne Felder angesprochen werden, dies ist wiederum bei der Analyse hilfreich, da dort beispielsweise das Vorhandensein von Feldern überprüft wird. Beim Ergänzen muss dann gegebenenfalls in Abhängigkeit der An- oder Abwesenheit eines Feldes ein Inhalt ergänzt werden. Die Anreicherung hat als Grundvoraussetzung wiederum die Möglichkeit, einzelne Felder hinzuzufügen. Die gefundenen Szenarien wurden in die vier Gruppen eingeteilt. Dort wurden diejenigen markiert, die besonders rele-

vant und repräsentativ erschienen, etwa, weil sie häufig genannt wurden. Für jede Gruppe wurden ein relativ einfaches Szenario und ein etwas komplexeres ausgewählt.

Das relativ einfache Szenario dient dazu, das Grundprinzip des Problems und dessen Lösung in KNIME beziehungsweise Catmandu zu zeigen. Mit dem komplexeren soll ein etwas speziellerer Fall vorgestellt werden. Dies soll helfen, die eigene Problemstellung zu abstrahieren und zu sehen, was das Grundproblem ist und wie es sich lösen lässt. Damit kann sich der Leser eventuell am Grundprinzip der Lösung orientieren. Eine Ausnahme ist dabei die Gruppe Anreicherung der Daten. Hier wurde nur ein Szenario bearbeitet, da es an sich schon recht komplex ist. Schließlich habe ich die Szenarien konkretisiert, etwa nach welchem Feld genau gefiltert werden soll. Anschließend mussten die Daten ausgewählt werden, mit denen die Szenarien bearbeitet werden konnten. Auch musste festgelegt werden, wie das Ergebnis des Szenarios aussehen sollte.

In der Beschreibung der Lösung soll darauf eingegangen werden, ob das Problem mit dem Tool und dem bisherigen Kenntnisstand lösbar war und wie viel Zeit dafür benötigt wurde. Auch soll beschrieben werden, wie viele Schritte (Nodes in KNIME beziehungsweise Fixes in Catmandu) benötigt werden. Es soll überprüft werden, ob in der Dokumentation passende Beispiele vorhanden sind, wie der Lösungsweg gefunden wurde und welche Probleme auf dem Weg dorthin auftauchten. Dabei soll auch auf Wege eingegangen werden, die nicht zum Ziel geführt haben, und warum dies der Fall war. Zum Schluss soll die jeweilige Lösung beschrieben werden und Screenshots beziehungsweise Programmcode abgebildet werden. Bei Catmandu wird zudem beschrieben, ob die Ergebnisse in KNIME und Catmandu dieselben waren. Die Konfiguration der Nodes in KNIME wird im Anhang B mittels Screenshots dokumentiert, damit die einzelnen Schritte nachvollzogen werden können.

Die Szenarien werden grundsätzlich mit MARCXML-Daten bearbeitet. Es sollten MARC-Daten verwendet werden, da diese im Bibliotheksbereich immer noch am dominantesten sind (Cole et al., 2018, S. 26). Laut Cole liegt das mitunter daran, dass viele Bibliotheksmanagementsysteme auf MARC basieren und viele Altdaten in MARC 21 vorliegen. Auch bei der Nutzung von Fremddaten nach dem RDA-Standard spielt MARC eine große Rolle (Cole et al., 2018, S. 26). Alle Daten, die vom BSZ zur Verfügung gestellt wurden, waren ebenfalls im MARC-Format. Da KNIME aber das binäre MARC-Format nicht kennt, da es doch sehr spezifisch für den Bibliotheksbereich ist, wurde auf MARCXML-Daten ausgewichen.

5 Analyse anhand der entwickelten Untersuchungskriterien

5.1 Plattformen und Aktualität

Beim Kriterium Plattformen und Aktualität soll überprüft werden, auf welchen Plattformen KNIME und Catmandu verfügbar sind und inwiefern diese gepflegt werden und Updates zur Verfügung stehen. Dies ist wichtig, um einschätzen zu können, ob die Programme auch auf den in der Bibliothek verfügbaren PCs verwendet werden können und um zu sehen, ob die Programme noch weiter verbessert und erweitert werden.

Dazu soll festgestellt werden, für welche Plattform Installationsmöglichkeiten bereitgestellt und beschrieben werden und wie diese gestaltet sind. Gibt es für jede der drei großen Plattformen (Windows, Linux und Mac) eine eigene Version? Wenn nein, welche Alternativen gibt es? Besteht beispielsweise die Möglichkeit zur Virtualisierung? Zusätzlich sollte untersucht werden, ob es eine Version oder Installationshinweise für die Nutzung auf einem Server gibt. Dies kann insbesondere dann von Nutzen sein, wenn sehr große Mengen an Daten verarbeitet werden sollen und die vorhandenen Serverstrukturen der Bibliothek genutzt werden sollen. Aufgrund der beschränkten Zeit und Ressourcen kann nicht getestet werden, ob die jeweiligen Installationsmöglichkeiten funktionieren, daher werden nur die in der Dokumentation beschriebenen Möglichkeiten genannt und im Hinblick auf Ubuntu beschrieben, welche Probleme bei der Installation auftraten.

Zudem soll erfasst werden, inwiefern die Software erweiterbar ist, um sie gegebenenfalls den eigenen Bedürfnissen anpassen zu können. Daher soll recherchiert werden, ob es Anleitungen gibt, um eigene Erweiterungen zu erstellen. Zudem soll erfasst werden, ob es Ansprechpartner für Vorschläge gibt oder externe Dienstleister, die für die Software Erweiterungen erstellen.

Um zu erfassen, ob die Software aktuell weiterentwickelt wird, soll untersucht werden, an wie vielen Tagen innerhalb des Zeitraums vom 01.09.2019 bis zum 31.10.2019 Commits in GitHub erstellt wurden. Zudem soll erfasst werden, von wann das aktuelle Release ist und ob es einen Rhythmus bei den Releases gibt. Auch die Häufigkeit der letzten Releases soll erfasst werden sowie der Abstand zwischen diesen. Diese Informationen sollen auf der Webseite und auf GitHub recherchiert werden.

5.1.1 KNIME

Die aktuelle Version für die KNIME Analytics Platform ist 4.0.2 („Download KNIME Analytics Platform“, o. J.). Für Windows gibt es drei Installationsmöglichkeiten: einen Installer, ein selbstextrahierendes Archiv und ein ZIP-Archiv. Alle drei Möglichkeiten gibt es jeweils in einer 64- und einer 32-bit-Version. Zudem gibt es KNIME für Linux (64-bit) und KNIME für Mac OSX (Version 10.11 oder höher – 64-bit) („Download KNIME Analytics Platform“, o. J.). Unter Ubuntu 18.04 mit GNOME gab es wie in Kapitel 3.1.2 beschrieben Probleme mit der Darstellung der Menüfenster. Daher musste auf die Windows-Version via KNIME gewechselt werden. Es ist positiv, dass es für alle drei großen Betriebssysteme eine Version gibt. Negativ fällt allerdings auf, dass die Linux-Version mit GNOME nicht benutzbar ist. In den FAQ fällt auf, dass es zwei Fragen gibt, die Probleme mit KNIME unter Windows beschreiben, eine zu Mac OS und fünf zu Linux („FAQ“, o. J.-a). Dies kann natürlich auch an der Heterogenität der Linux-Welt liegen. Alternativen zur Virtualisierung werden von KNIME selbst scheinbar nicht bereitgestellt, was jedoch theoretisch auch nicht nötig sein sollte, da es ja für alle drei großen Systeme eine Version gibt.

Es gibt eine KNIME-Version für Server („KNIME Server“).Für diese muss allerdings eine Jahreslizenz oder ein voreingerichteter Server bei Azure oder AWS gekauft werden („KNIME Server“, o. J.). Dass die Möglichkeit besteht, KNIME auf einem Server zu nutzen, ist positiv, jedoch könnten die Kosten ein Problem für Bibliotheken darstellen.

Es ist möglich, die Funktionalität von KNIME durch Erweiterungen anzupassen. Dabei gibt es Erweiterungen von KNIME selbst, von der Community erstellte Open-Source-Erweiterungen und Erweiterungen von Partnern von KNIME („Install Extensions and Integrations“, o. J.). Zudem wird auf der Webseite ein Quickstart Guide für das Erstellen eigener Erweiterungen bereitgestellt („Create a New KNIME Extension: Quickstart Guide“, o. J.). Außerdem gibt es auf der Webseite eine Sektion mit Informationen für Entwickler („Developers“, o. J.) und eine weitere mit Services für Community-Entwickler („Instructions for Developers“, o. J.). Auch im Forum gibt es einen Bereich für Entwickler („KNIME Community Forum“, o. J.). Diese Möglichkeiten und Hilfestellungen sind als positiv zu bewerten.

Wenn man eine neue Erweiterung benötigt und diese nicht selbst programmieren kann, verweist KNIME auf sein Partnernetzwerk. Manche Partner entwickeln auch im Auftrag Erweiterungen („KNIME Trusted Partners“, o. J.). Ausnahmen werden von der KNIME AG gemacht, wenn eine Firma dringend ein Feature benötigt, das gerade keine hohe Priorität hat. Dann entwickelt die KNIME AG eventuell das Feature und lässt sie sich gegebenenfalls für die Entwicklung bezahlen. Das Ergebnis macht die KNIME AG dann aber meist Open Source, damit auch andere davon profitieren können („KNIME Open Source Story“, o. J.). Diese Lösung scheint geeignet, die Informationen dazu finden sich aber nicht direkt. Auch wird es als kleine Einrichtung vermutlich schwierig, die Entwicklung zu bezahlen. Vielleicht wäre es eine Option, eine Möglichkeit zu bieten, um

Vorschläge einzureichen oder um festzustellen, welche Features gerade von vielen benötigt werden.

Im Zeitraum vom 01.09.2019 bis zum 31.10.2019 wurden an 28 Tagen Commits in knime-core auf GitHub eingefügt („knime/knime-core“, 2019). Das letzte Release ist KNIME 4.0.2 und wurde auf GitHub am 30.09.2019 erstellt, das offizielle Releasedatum ist eventuell ein bis zwei Tage später („knime/knime-core“, o. J.). Insgesamt gibt es alle ein bis zwei Monate ein Release. Circa jedes halbe Jahr (meist im Dezember und im Juni oder Juli) erfolgt der Schritt zur nächsten Versionsnummer (von 3.5 auf 3.6 beispielsweise) („Previous Versions“, o. J.). Das Tool wird also durchaus aktiv weiterentwickelt und es gibt auch einen Rhythmus, in dem neue Versionen erscheinen.

5.1.2 Catmandu

Bei Catmandu gibt es keine dezidierten Versionen für jede Plattform. Dies liegt vermutlich an der UNIX-Basis, die eine Portierung zu Windows erschwert. Für OSX gibt es eigene Installationshinweise, ebenso für verschiedene Linux-Distributionen. Für Windows werden die Optionen Docker, Strawberry Perl (Catmandu, o. J.-a) und an anderer Stelle VirtualBox genannt („Download“, o. J.), dies wird unter dem nächsten Punkt Alternativen ausführlicher beschrieben. Jedoch ist auch mit ein wenig UNIX-Erfahrung und Zugriff auf einen UNIX-Rechner die Dokumentation verwirrend, da viele verschiedene Wege beschrieben werden und die Vor- und Nachteile nicht klar sind und auch keine direkte Empfehlung gegeben wird. Zudem steht am Anfang ein Installationsweg via CPAN ohne wirkliche Hinweise, für wen und wann das geeignet ist oder ob dieser Weg empfohlen wird. Danach folgen Hinweise für verschiedene Distributionen. Dass es keine Windows-Version gibt, ist zwar für dessen Nutzer schade, aber aufgrund der Unterschiede verständlich. Zusätzlich ist aber die Anleitung sehr knapp und verwirrend, wenn man nicht weiß, womit man es zu tun hat. Mit Mac OSX und Linux kann man Catmandu ohne zusätzliche Schicht anwenden.

Im Kapitel Installation der Dokumentation wird für „Windows, Mac OSX, Linux“ beschrieben, dass es mit jedem Release ein Docker Image gibt, das genutzt werden kann. Es wird auf die Dokumentation von Docker zur Installation von diesem verwiesen. Zudem wird beschrieben, dass/wie Strawberry Perl genutzt werden könne(Catmandu, o. J.-b). Auf dem Catmandu-Blog wird unter „Download“ auf die Installationshinweise in der Dokumentation verwiesen, wenn man mit UNIX vertraut ist und Zugriff auf einen UNIX-Rechner hat. Für Menschen, die damit nicht vertraut sind, gäbe es eine VirtualBox und Installationshinweise, mit der die IT-Abteilung das einrichten könne. Immerhin scheint die VirtualBox noch aktualisiert zu werden, unter last updated beim Image steht 2019-04-09 („Download“, o. J.). Hier stellt sich die Frage, warum die Virtualisierungsmöglichkeit mit VirtualBox an einer anderen Stelle steht und nicht in der Dokumentation genannt wird. Für jemanden, der mit der Kommandozeile gar nicht vertraut ist, ist die vorhandene Dokumentation auf der Webseite und auf GitHub sehr

knapp. Andererseits ist es auch schwer und außerhalb des Rahmens einer solchen Dokumentation, das komplett einzuführen. Dennoch sollte es Hinweise geben, wie man sich weiter informieren kann. Auf dem Blog gibt es mit dem Adventskalender, der in die Benutzung der Kommandozeile und von Catmandu einführt, eine ganz gute Möglichkeit, einen leichten Einstieg zu gestalten.

Bei den Installationshinweisen finden sich auch welche für Ubuntu Server 12.04.4 LTS (Catmandu, o. J.-a). Die Hinweise wirken sehr knapp (es handelt sich nur um eine Liste von nötigen Befehlen), aber immerhin ist es vorhanden. Die Nutzung auf Servern mit anderen Betriebssystemen als Linux wird nicht beschrieben, spielt vermutlich aber auch eine geringere Rolle.

Es gibt eine Anleitung, wie man sich eine Entwicklungsumgebung für Catmandu einrichten kann, sowie Hinweise, wie die Zusammenarbeit auf GitHub ablaufen kann (Catmandu, o. J.-e).

Laut der Dokumentation gibt es eine Liste namens „missing modules“, in der Ideen und Ressourcen für neue Module gesammelt werden und die man gerne ergänzen kann („LibreCat/Catmandu“, 2016a). Sie scheint allerdings nicht sonderlich gepflegt oder genutzt zu werden, da die letzte Änderung vom 08.04.2016 ist. Zudem führt der Link zur Liste zurück auf die Seite der Dokumentation, wenn man sich die Dokumentation auf der Webseite von LibreCat ansieht, statt auf GitHub. Das scheint daran zu liegen, dass die Dokumentation auf der Webseite und auf GitHub verknüpft ist. Der Link zur Liste funktioniert aber nur von der Dokumentation auf GitHub aus. Es ist positiv, dass es eine offen einsehbare und ergänzbare Liste gibt, aber sie scheint nicht mehr genutzt zu werden und der kaputte Link von der Webseite sorgt für Verwirrung.

Externe Dienstleister für Erweiterungen werden keine genannt, was aber bei der Größe und Struktur des Open-Source-Projektes auch nicht zu erwarten war. Im Zeitraum vom 01.09.2019 bis zum 31.10.2019 gab es an fünf Tagen Commits auf GitHub („LibreCat/Catmandu“, 2019a). Das letzte Release ist Version 1.2009 vom 04.11.2019 („LibreCat/Catmandu“, o. J.). Das Projekt wird also noch aktiv weiterentwickelt. Ein wirklicher Rhythmus ist bei den Releases nicht zu erkennen („LibreCat/Catmandu“, o. J.). Sie werden vermutlich in Abhängigkeit davon veröffentlicht, ob es wesentliche Änderungen gab. Auch wenn dann manchmal ein paar Monate nichts passiert, ist doch eine kontinuierliche Arbeit an Catmandu zu erkennen. Die Abstände zwischen den kleineren Sprüngen (zum Beispiel von 1.2004 auf 1.2005) sind sehr unterschiedlich. Manchmal gibt es am selben Tag oder innerhalb von wenigen Tagen zwei Releases, dann aber auch mal fast drei Monate keines. Auch bei den größeren Sprüngen (von 1.08 auf 1.09 etwa) sind die Abstände sehr unterschiedlich („LibreCat/Catmandu“, o. J.).

5.2 Support und Dokumentation

Als zweites Kriterium soll untersucht werden, welche Formen von Support und Dokumentation vorhanden sind, wie sie gefunden werden können und welche Qualität sie haben. Dies ist wichtig für neue Nutzer, damit sie einen möglichst einfachen Einstieg in das neue Programm erhalten. Auch bereits erfahrene Nutzer, die eine für sie neue Funktion nutzen möchten, brauchen eine übersichtliche, gut verständliche Dokumentation. Außerdem soll festgestellt werden, ob und in welchem Rahmen die Möglichkeit besteht, Hilfe bei Problemen zu bekommen, die man nicht alleine mit der Dokumentation lösen kann. Dies wird unter dem Begriff Support zusammengefasst.

Um den Bereich Support analysieren zu können, soll überprüft werden, ob eine Mailingliste vorhanden ist, ob dort auf Fragen geantwortet wird und ob auch das Entwicklerteam dort aktiv ist. Ebenso soll überprüft werden, ob ein Forum vorhanden ist, und ob dort Aktivität seitens anderer Nutzer oder der Entwickler besteht. Zudem soll überprüft werden, ob es externe Dienstleister gibt, die für Support zur Verfügung stehen.

Um den Bereich Dokumentation analysieren zu können, soll ein Überblick gegeben werden, auf welchen Webseiten Teile der Dokumentation auffindbar sind und inwiefern die Seiten aufeinander verweisen. Anhand dieser Informationen soll eingeschätzt werden können, ob die jeweilige Dokumentation von verschiedenen „Startpunkten“ aus auffindbar ist und ob sie gebündelt an einem Ort liegt oder ob man eventuell lange suchen muss, bis man die richtige Seite findet. Dabei soll auch darauf geachtet werden, ob die Dokumentation sich wiederholt. Weiterhin ist es wichtig, zu überprüfen, ob die Elemente des Programms beschrieben werden (die Nodes bei KNIME und die Importer/Fixes/etc. bei Catmandu). Dazu gehört auch, ob die verschiedenen Erweiterungen und Module beschrieben werden.

Weiterhin soll überprüft werden, ob für typische Prozesse Anleitungen vorliegen und ob es Cheat Sheets gibt, um sich einen schnellen Überblick zu verschaffen. Dabei ist insbesondere zu untersuchen, ob nachvollziehbar ist, was in dem Beispiel gemacht wird und ob man das Beispiel findet. Dazu soll überprüft werden, ob dem Beispiel eine Beschreibung vorangestellt wird und ob die einzelnen Schritte etwa durch Kommentare erklärt werden oder nur das jeweilige Kommando beziehungsweise der Node zu sehen ist. Wichtige Fragen sind hier auch: Wie komplex sind die Beispiele? Wird auf die benötigten Grundlagen verwiesen? Das Auffinden des Beispiels kann hier nicht direkt getestet werden, bei der Bearbeitung der Szenarien im nachfolgenden Kapitel wird aber darauf eingegangen, ob es entsprechende Beispiele gab und ob diese vor der Bearbeitung des Szenarios gefunden wurden.

Um Anfängern den Einstieg möglichst einfach zu machen, bieten viele Programme spezielle Hilfsmittel für diese an. Deswegen soll überprüft werden, ob dies auch bei KNIME und Catmandu der Fall ist, um was es sich dabei handelt, welches Vorwissen

dabei vorausgesetzt wird und wo diese Ressourcen sich befinden, denn hier ist besonders wichtig, dass diese leicht zu erkennen und finden sind.

5.2.1 KNIME

Bei KNIME scheint es keine Mailingliste zu geben, dafür existiert ein Forum. Es wird recht viel gepostet und auf die meisten Posts antwortet jemand. Außerdem gehen verschiedene Personen auf Fragen im Forum ein und sind dabei als „KNIME Team Member“ markiert („Wrong results from Item Set Finder / Association Rule Learner“, 2018). Zusätzlich bieten externe Dienstleister Support an. Diese gehören auch zum KNIME Partner Programm („KNIME Trusted Partners“, o. J.).

Insgesamt scheint die gesamte Dokumentation auf der KNIME-Webseite zu liegen, was ein großer Pluspunkt ist. Es gibt zwar noch einen YouTube-Kanal, aber dessen Videos lassen sich auch in die Webseite einbinden. Dafür ist die Struktur der Webseite natürlich entsprechend komplex. Es kommt dennoch vor, dass man etwas sucht, was man schon hatte, und es nicht findet, weil man an der falschen Stelle sucht.

Die Seitenstruktur mit der relevanten Dokumentation lässt sich wie folgt abbilden:

- „Get Started“ https://www.knime.com/next-steps
- Beispiel-Workflows aus verschiedenen Geschäftsbereichen https://www.knime.com/solutions
- KNIME Partners https://www.knime.com/partners
- KNIME Hub https://hub.knime.com/ (Nodes, Components, Workflows, Extensions Suchmaschine)
- einen Blog https://www.knime.com/blog
- Forum https://forum.knime.com/
- Events https://www.knime.com/learning/events
- Learning https://www.knime.com/resources
    - Getting started https://www.knime.com/knime
    - FAQ https://www.knime.com/faq
    - Learning Hub https://www.knime.com/learning-hub mit Auswahl an Materialien für verschiedene Zwecke (positiv)
    - Data Science E-Learning Course https://www.knime.com/knime-introductory-course
    - KNIME Server E-Learning Course https://www.knime.com/knime-server-e-learning-course
    - KNIME Dokumentation https://docs.knime.com/

- Informationen für Entwickler https://www.knime.com/developers
- KNIME Cheat Sheets https://www.knime.com/learning/cheatsheets
- White Papers https://www.knime.com/white-papers
- KNIME Press (E-Books / PDFs zum kaufen) https://www.knime.com/knime-press
- KNIMETV auf YouTube
- Changelog https://www.knime.com/changelogs

Teilweise sind die Unterseiten von verschiedenen Stellen aus verlinkt. Wie man sieht, ist es insgesamt sehr viel, sodass man oft nicht weiß, wo man anfangen soll.

Auf der GitHub-Hauptseite wird in der Einführung auf die KNIME-Webseite verwiesen, außerdem noch auf das Forum und auf Repositorien, die genutzt werden können, um sich eine eigene Entwicklungsumgebung für Erweiterungen einzurichten („knime/knime-base“, 2019). Auch in der „About“-Sektion des YouTube-Kanals („KNIMETV“, o. J.) wird auf die KNIME-Webseite verlinkt.

Mit dem KNIME Hub („KNIME Hub“, o. J.) kann nach Workflows, Erweiterungen und Nodes recherchiert werden. Die einzelnen Elemente werden dort auch beschrieben und Hinweise zur Benutzung gegeben. Teilweise sind die Beispiele zu einfach, etwa bei den XML-Nodes. Die Fallstricke bei der Ansprache werden dabei nicht offensichtlich und da auch die Node-Beschreibung nicht groß weiter hilft, muss man sich mit anderen Ressourcen außerhalb von KNIME behelfen. Meist sind die Beispiele (zumindest die auf dem Example-Server von KNIME) mit einer kurzen Beschreibung ausgestattet, was gemacht werden soll. Die einzelnen Nodes und insbesondere deren Konfiguration ist tendenziell eher nicht beschrieben. Cheat Sheets sind auf der Webseite verfügbar („KNIME Cheat Sheets“, o. J.).

Für Anfänger gibt es verschiedene Ressourcen:

- „Get Started“ https://www.knime.com/next-steps (darauf wird man nach dem Runterladen des Programms gelenkt)
- Man kann sich für E-Mails eintragen, die einen dann durch die ersten Tage mit KNIME leiten sollen
- „Getting Started“ unter Learning https://www.knime.com/knime (Links zu Download, Install, Build a workflow, Example workflows, Install Extensions, More Resources)
- ein Artikel „Seven things to do after installing KNIME Analytics Platform“ https://www.knime.com/blog/seven-things-to-do-after-installing-knime-analytics-platform

Das sind tendenziell fast zu viele Ressourcen und es ist etwas unübersichtlich, was dazu verleitet, das Programm einfach zu testen und die Beispiele anzusehen.

5.2.2 Catmandu

Für Catmandu gibt es eine Mailingliste mit der Adresse librecat-dev(at)lists.uni-bielefeld.de (librecat-dev, o. J.). Auf der Mailingliste wird nach Einsicht des offenen Archivs auf Fragen geantwortet, auch vom Entwicklerteam (Steenlant, 2019c). Ein Forum und externe Dienstleister für Support sind nicht vorhanden. Dafür ist das Projekt vermutlich nicht groß genug und auch die Mailingliste ist nicht sehr stark frequentiert („The librecat-dev Archives“, o. J.).

Auf der Webseite („LibreCat & Catmandu“, o. J.) gibt es unter anderem die Reiter „Documentation“, „Distributions“, „Cheat Sheet“ und „Blog“. Der Reiter Documentation führt zu einer Unterseite, auf der die Inhalte des GitHub Wikis auf einer Seite dargestellt werden. Der Reiter Distributions führt zu einer Liste aller Distributionen. Diese werden mit ihrer metacpan-Seite verlinkt, auf der sich weitere Informationen zu den einzelnen Distributionen befindet. Der Reiter Cheat Sheet führt zu einem PDF mit vier Seiten, auf der verschiedene Befehle zusammengestellt sind (Hochstenbach, 2019c). Der Reiter Blog führt zum Blog, auf dem weitere Dokumentation zu finden ist, insbesondere ein Tutorial in Form eines Adventskalenders für Einsteiger.

Der Adventskalender („Tutorial“, 2019) führt in 18 Tagen in die Benutzung der Kommandozeile und in Catmandu ein. Er ist sehr hilfreich, da er Schritt für Schritt aufeinander aufbaut und Screenshots eingebunden sind, mithilfe derer der Output des Programms überprüft werden kann. Mit dem Adventskalender war der Einstieg sehr viel leichter als mit der allgemeinen Dokumentation. Die einzelnen Schritte und auch einige UNIX-Grundlagen werden erklärt. Um vom Blog auf die Webseite von LibreCat zu kommen, muss man auf den Reiter „About“ klicken, dort ist die Seite verlinkt. Zu GitHub gibt es keinen offensichtlichen Link, ebenso nicht zu metacpan.

Auf metacpan.org sind auf den Seiten der verschiedenen Distributionen und Module Anleitungen zu diesen zu finden. Da diese jeweils an einen Entwickler angeschlossen sind, scheint es keine Übersicht aller Distributionen und Module von Catmandu zu geben, auch wird von den jeweiligen Seiten nicht zu anderen Elementen der Dokumentation verlinkt. Nur von dem Modul Catmandu::Introduction wird zur LibreCat-Webseite verlinkt. Wenn man allerdings bei einem Modul ist, dessen Entwickler nicht Steenlant ist, findet man diese Einführung eventuell nicht.

Auf GitHub („LibreCat/Catmandu“, 2019b) gibt es eine kurze Einführung auf der Hauptseite sowie ein Wiki. Das Wiki wird auf der LibreCat-Webseite als Basis für den Reiter „Dokumentation“ verwendet. In der kurzen Einführung auf der GitHub-Hauptseite wird verlinkt auf die LibreCat-Webseite mit der dazugehörigen Dokumentation, auf den Blog und das Adventskalender-Tutorial.

Gerade für neue Benutzer ist es sehr schwierig, sich einen Überblick über die verschiedenen Elemente zu machen. Das kann dazu führen, dass zwar Dokumentation zu einem bestimmten Thema vorhanden ist, diese aber nicht gefunden wird. Die Webseite und auf GitHub wird auf die meisten anderen Ressourcen verlinkt. Jedoch wird nicht vom dem Blog und von metacpan (was durch dessen Struktur allerdings auch schwierig ist) auf die anderen Ressourcen verlinkt. Dennoch führt es dazu, dass einzelne Inhalte oft wiederholt werden, während andere schwierig aufzufinden sind.

Auf metacpan gibt es eigentlich für jedes Modul eine Beschreibung. Dort sind auch die Erweiterungen beschrieben. In der allgemeinen Dokumentation werden ebenfalls verschiedene Elemente, aber nicht alle beschrieben. Zudem gibt es auf metacpan ein MARC-Tutorial („Catmandu::MARC::Tutorial“, o. J.), bei dem verschiedene übliche Prozesse beschrieben werden. Bei den Beispielen sind teilweise Beschreibungen vorangestellt, teilweise nicht. Einzelne Schritte sind manchmal, aber eher selten verlinkt. Daher ist oft nicht ganz verständlich, was eigentlich gemacht wird oder was Ergebnis des Kommandos ist. Die Ziele der Beispiele sind nicht zu komplex, aber die Beispiele sind zu knapp, um sie zu verstehen. Oft war auch nicht klar, was in dem Beispiel jetzt zur Syntax gehört oder zum Feldname. Zusätzlich war unklar, bei welcher Feldstruktur welche Syntax verwendet werden muss, etwa bei Arrays. Die Begriffe werden nicht immer einheitlich verwendet. Es wird eher weniger auf Grundlagen verwiesen, nur bei der Dokumentation speziell für Anfänger. Oft ist nur ein Teil des Befehls angegeben, der benötigt wird, den Rest muss man sich dann aus verschiedenen Quellen zusammensuchen.

5.3 Datenformate und Metadatenstandards

Ein weiteres wichtiges Element bei der Arbeit mit Metadaten sind verschiedene Datenformate und Metadatenstandards. Es existiert eine Vielzahl an verschiedenen Formaten und Standards, beispielsweise MARC, MAB, CSV, YAML, JSON, XML, METS, MODS, Dublin Core oder TEI. Bibliotheken sehen sich einer zunehmenden Fülle an Datenquellen, etwa dem Verbund oder einem Konsortium, ausgesetzt. Zudem gibt es mehr als einen Datenempfänger, wie das Bibliotheksmanagementsystem oder das Resource Discovery System (Pfeffer, 2016, S. 5). Diese Vielzahl an Systemen führt auch zu einer immer größeren Anzahl an Datenformaten und Schemata. Nun soll anhand einzelner Beispiele untersucht werden, ob KNIME und Catmandu mit den verschiedenen Formaten und Schemata umgehen können und somit Bibliothekare bei ihrer Arbeit mit Daten in diesen Formaten unterstützen können.

Zu Beginn der Arbeit wurden einzelne Standards und Formate vorgestellt (MARC 21, MARCXML, RDF und Dublin Core). Nun soll untersucht werden, inwiefern diese von den Programmen KNIME und Catmandu unterstützt werden. Exemplarisch ausgewählt wurden MARC und MARCXML, da sie im Bibliotheksbereich sehr weit verbreitet sind

und viele Daten in diesem Format vorliegen oder in diesem Format erstellt werden (Cole et al., 2018, S. 26). Daher war es am wichtigsten, KNIME und Catmandu in Hinblick auf diese beiden Formate zu testen. Zusätzlich wurde RDF ausgewählt, da es ein Standard ist, der im Zuge von Linked Data und dem Semantic Web zunehmend an Bedeutung gewinnt (Hipler et al., 2018, S. 166). Auch die DNB veröffentlich ihre Daten inzwischen zusätzlich als RDF („Linked-Data-Service“, o. J.). Dublin Core wurde ausgewählt, da es sehr weit verbreitet ist (Cole et al., 2018, S. 27).

Um analysieren zu können, inwiefern die Formate unterstützt werden, soll recherchiert werden, ob diese eingelesen werden können. Zudem soll genannt werden, welche spezifischen Funktionen (Nodes beziehungsweise Module) es in KNIME und Catmandu gibt, die die Arbeit mit dem Format oder Standard erleichtern. Ein weiterer Aspekt ist, ob es passende Beispiele gibt, die den Umgang mit dem jeweiligen Format veranschaulichen. Zuletzt soll durch Recherchen geklärt werden, ob die Möglichkeit besteht, von dem einen in das andere Format umzuwandeln. Die Informationen stützen sich dabei größtenteils auf die Dokumentation, eventuell auch auf die im Programm sichtbaren Funktionen (bei der grafischen Oberfläche von KNIME).

5.3.1 KNIME

Man kann .mrc- und .mrk-Dateien zwar mit dem File Reader einlesen, aber sie danach nicht wirklich sinnhaft bearbeiten, da die Struktur fund die Ansprechbarkeit von einzelnen Feldern fehlt.

XML kann von KNIME verarbeitet werden, dazu gibt es einige Nodes. Daher kann auch MARCXML verarbeitet werden. Es gibt eine Erweiterung namens KNIME XML-Processing („KNIME XML-Processing“, o. J.). Diese enthält die Nodes Column to XML, String to XML, XML Column Combiner, XML Reader, XML Row Combine and Write, XML Row Combiner, XML Writer, XPath und XSLT. Dabei sind Workflows verlinkt, in denen die Nodes zur Anwendung kommen. Auch für die einzelnen Nodes gibt es Seiten mit kurzen Beschreibungen des Nodes, der Ports und Optionen („KNIME XML-Processing“, o. J.). Im Node Repository sind die XML-Nodes unter Stuctured Data → XML zu finden. Man kann die üblichen XML-Methoden verwenden, um mit Daten im MARCXML-Format zu arbeiten.

In KNIME gibt es eine Erweiterung namens „KNIME Semantic Web“, die verschiedene Funktionen liefert, mit denen auch RDF-Daten verarbeitet werden können („KNIME Semantic Web“, o. J.). Die Nodes heißen: Memory Endpoint, SPARQL Endpoint, Virtuos Endpoint, Triple File Reader, Triple File Writer, SPARQL File Inserter, SPARQL File Writer, SPARQL Delete, SPARQL Executor, SPARQL Insert, SPARQL List Graph Names, SPARQL Query und SPARQL Update. Auch hier sind wieder Workflows verlinkt und zu den einzelnen Nodes gibt es Beschreibungsseiten („KNIME Semantic Web“, o. J.). Im Node Repository sind die Nodes unter KNIME Labs → Semantic Web/Linked Data zu finden, die Erweiterung musste erst installiert werden.

Da Dublin Core vorrangig als XML, RDF oder HTML serialisiert wird („DCMI: Recommendation“, 2019), sollte es möglich sein, Dublin Core-Metadaten mit KNIME zu bearbeiten. Es wurden jedoch keine spezifischen Nodes für Dublin Core gefunden.

Es ist nicht direkt ersichtlich, inwiefern Daten von dem einen in das andere Format umgewandelt werden können. Vermutlich müssen die Daten von Hand dafür aufbereitet werden, damit die richtigen Felder zusammengeführt werden oder die Form dem Zielformat entspricht.

5.3.2 Catmandu

In Catmandu gibt es verschiedene Module, um mit MARC-Daten zu arbeiten. Diese sind zusammengefasst in der Distribution Catmandu-MARC-1.254 (Hochstenbach, 2019f). Einige der Module befassen sich speziell mit MARCXML. So gibt es einen Exporter und einen Importer für MARCXML sowie einen Fix, mit dem MARC in MARCXML umgewandelt werden kann (Hochstenbach, 2019f).

Für MARC allgemein gibt es hauptsächlich Module aus den Bereichen Importer, Exporter und Fixes. Die Importer und Exporter bieten die Möglichkeit, verschiedene MARC-Formate zu importieren beziehungsweise zu exportieren. Diese Formate umfassen beispielsweise USMARC, MARCXML, MARC als JSON, MARC nach ISO oder MARC als das sequentielle Format von Ex Libris‘ Aleph (Hochstenbach, 2019f). Bei den Fixes gibt es verschiedene Möglichkeiten, das Vorhandensein oder den Inhalt eines MARC-(Unter)Feldes zu überprüfen. Man kann Felder neu hinzufügen, entfernen oder auf einen bestimmten Wert setzen. Zudem können Felder kopiert und eingefügt werden oder der Inhalt eines Feldes einem neuen Feld zugewiesen werden. Bei der Verwendung von MARC-spezifischen Fixes kann der Pfad zum jeweiligen Feld einfach durch den Tag und gegebenenfalls den Code angegeben werden. Also kann man beispielsweise durch marc_map(245a,title) den Inhalt des Feldes 245 Unterfeld a in das neue Feld ‚title‘ kopieren. Durch ein Modul aus dem Bereich Validator besteht außerdem die Möglichkeit, MARC-Records zu validieren, indem sie mit dem MARC 21-Schema abgeglichen werden (Hochstenbach, 2019f).

Alle MARC-Formate können auf dieselbe Weise verarbeitet werden, es können also auch dieselben Fixes angewendet werden. Es muss lediglich die genutzte Art von MARC in der Form „MARC --type ISO“ mit angegeben werden, also beispielsweise

```
catmandu convert MARC --type XML to YAML < Beispieldatei.mrc.xml
```

Mit dieser Angabe können zudem die jeweiligen MARC-Formate ineinander oder auch in ein anderes, von Catmandu unterstütztes, Format umgewandelt werden (Hochstenbach, 2019e).

In der Übersicht zu der Distribution Catmandu-MARC-1.254 (Hochstenbach, 2019f) sind alle Hilfsseiten zu den Importern, Exportern, Fixes und dem Validator verlinkt. Auf diesen finden sich Beispiele, wie das jeweilige Modul auf der Kommandozeile oder in

einem PERL-Skript angewandt werden kann, zudem wird eine knappe Beschreibung gegeben und, falls vorhanden, die Optionen erklärt.

Zu RDF gibt es die Distribution Catmandu-RDF-0.32. Diese Distribution enthält einen Importer, einen Exporter und zwei Fixes (Voß, 2017e). Mit dem ersten Fix, Catmandu::Fix::aref_query, kann bei aREF-kodierten RDF-Daten der Inhalt eines Elements in ein neues Feld kopiert werden (Voß, 2017b). Mit dem anderen Fix, Catmandu::Fix::rdf_ldf_statements, können Werte als Abfragen an Linked Data Fragments-Endpunkte geschickt werden, wobei das Ergebnis der Abfrage den gesendeten Wert ersetzt (Voß, 2017c). Mit dem Importer können RDF-Daten in verschiedenen Formaten, wie beispielsweise „RDF/XML, RDF/JSON, Turtle, NTriples“ (Voß, 2017d), eingelesen werden oder vom Endpunkt einer SPARQL-Anfrage eingespielt werden (Voß, 2017d). Mit dem Exporter können Daten als RDF ausgegeben werden. Dabei sind verschiedene Serialisierungen möglich, etwa auch Notation3 (Voß, 2017a). Wie bei der MARC-Distribution sind die Hilfsseiten zu den jeweiligen Modulen verlinkt. Auf diesen befinden sich wieder Beispiele, eine Beschreibung und eventuelle Optionen, Methoden, etc.

Zur Umwandlung schreibt Voß (2017d), dass für die Umwandlung von einer RDF-Serialisierung in eine andere RDF-Serialisierung spezifische Programme für RDF wahrscheinlich geeigneter sind. Vorteil von Catmandu::RDF sei aber, dass es für Umwandlungen zwischen RDF und einem anderen Format geeigneter sei (Voß, 2017d).

Da Dublin Core vor allem als XML, RDF oder HTML serialisiert wird, sollte Catmandu damit umgehen können. Es wird auch explizit genannt (Catmandu, o. J.-a), eventuell sind also Mappings hinterlegt, es gibt aber keinen Exporter oder ähnliches. Daher bleibt unklar, inwiefern speziell Dublin Core, unabhängig von der generellen Möglichkeit, XML und RDF zu verwenden, unterstützt wird.

Im Adventskalender, der den Einstieg in Catmandu erleichtern soll, ist eine Anleitung („Day 15“, 2014), um von MARC zu Dublin Core umzuwandeln. Dabei muss man selbst die Felder mappen. Allerdings wird dort auch die ISBN und ISSN gemappt mit dem Hinweis, dies seien noch nicht die Dublin Core-Felder (es gibt kein Element ISBN oder ISSN), diese würden später bearbeitet werden. Jedoch endet die Beschreibung des Verfahrens kurz darauf. Im *Cookbook* („LibreCat/Catmandu“, 2016b) wird hingegen erläutert, wie Dublin Core-Metadaten mittels OAI-PMH eingelesen oder in YAML und CSV umgewandelt werden können – hier wird nichts gemappt, nur eingelesen. Der Catmandu-Store FedoraCommons arbeitet standardmäßig mit Dublin Core (Hochstenbach, 2018b).

5.4 Schnittstellen und Datenbanken

Als viertes Untersuchungskriterium werden Schnittstellen und Datenbanken betrachtet. Die Möglichkeit zur Nutzung von Schnittstellen durch Bibliotheken ist wichtig, damit

schnell und ohne Zwischenschritte Daten aus einer externen Datenquelle in den eigenen Bestand geladen werden können. Da Bibliotheken selbst mit Datenbanken arbeiten, sollte es auch möglich sein, diese aus dem Programm heraus anzusteuern.

Bei diesem Untersuchungskriterium soll beispielhaft die Unterstützung der eingangs vorgestellten Schnittstellen OAI-PMH und REST sowie der Datenbank MongoDB analysiert werden. OAI-PMH wurde ausgewählt, da es sich dabei um eine Schnittstelle speziell für *digital libraries* (Cole & Foulonneau, 2007, S. 3) und Repositorien (OAI, o. J.) handelt. Beides sind übliche Tätigkeitsbereiche von Bibliotheken, daher ist OAI-PMH ein wichtiger Standard. Zudem ist OAI-PMH ausgelegt für strukturierte Metadaten (OAI, o. J.), wie sie in Bibliotheken vorrangig vorkommen (Cole et al., 2018, S. 8). Wie beschrieben werden sowohl OAI-PMH als auch REST-Schnittstellen von vielen Anwendungen im Bibliotheksbereich unterstützt, etwa von der Repositoriensoftware Fedora, von der Digital Public Library of America oder der Open Source-Bibliotheksmanagementsystem Koha („APIs and protocols supported by Koha“, 2019; „Features - Fedora“, o. J.; Karadkar et al., 2016; Mitchell, 2015, S. 184). Während OAI-PMH aus dem Bereich Eprints stammt und damit ausgelegt ist auf bibliografische Metadaten, wie sie in Bibliotheken vorkommen (Mitchell, 2015, S. 177), wurde REST als Beispiel gewählt, da es sich um einen allgemeineren Architekturstandard handelt, der breit unterstützt wird („RESTful Webservices (1)“, o. J.).

Als Vertreter für die Nutzung von Datenbanken in Bibliotheken wurde MongoDB gewählt. In einem Artikel über die Speicherung bibliografischer Metadaten in verschiedenen Datenbankmodellen beschreiben Stephan und Klettke (2017), dass JSON-basierte Systeme bei größeren Datenmengen Vorteile hätten (Stephan & Klettke, 2017, S. 1159). Insgesamt scheint JSON als Basis recht gute Eigenschaften aufzuweisen und-MongoDB wird dabei als Vertreter für diese Systeme untersucht (Stephan & Klettke, 2017, S. 1159). Allgemein ist MongoDB unter den dokumentenorientierten Datenbanken am weitesten verbreitet (solid IT, 2019).

Zuerst soll überprüft werden, ob das Programm die jeweilige Schnittstelle beziehungsweise Datenbank unterstützt, das heißt ob es in KNIME Nodes gibt, die genutzt werden können, und ob es in Catmandu Importer oder Exporter Module gibt. Im Anschluss soll geprüft werden, ob eine Anleitung und Beispiele zur Verfügung stehen, mit denen die Nutzung erlernt werden kann.

5.4.1 KNIME

Die Suche nach „OAI-PMH“ im KNIME Hub ergibt keine Ergebnisse, die auf das gewünschte zutreffen. Vielleicht ist es aber möglich, eine OAI-PMH-Schnittstelle über REST anzusprechen. Bei Drupal gibt es beispielsweise auch ein Modul, mit dem OAI-PMH mittels REST angesprochen werden kann („REST OAI-PMH“, 2019).

Für REST gibt es eine „KNIME REST Client Extension“, die vier Nodes enthält, um mit REST-Schnittstellen zu interagieren. Diese sind DELETE Request, GET Request, POST Request und PUT Request („KNIME REST Client Extension“, o. J.). Im Node Repository innerhalb von KNIME werden die Nodes unter „Tools & Services“ als „REST Web Services“ angegeben.

Die „KNIME MongoDB Integration“ Erweiterung bietet fünf Nodes, um mit einer MongoDB zu interagieren: MongoDB Reader, MongoDB Remove, MongoDB Save, MongoDB Update sowie MongoDB Writer („KNIME MongoDB Integration“, o. J.). Im Node Repository innerhalb von KNIME werden die Nodes dann allerdings nicht unter DB (Datenbanken) angezeigt, sondern unter KNIME Labs und dort unter MongoDB. Die Erweiterung musste erst installiert werden.

Die Nodes zu MongoDB und zu REST sind auf der jeweiligen Beschreibungsseite der Erweiterung mit ihren Hilfeseiten verlinkt. Dort wird eine kurze Beschreibung gegeben, die Ports des Nodes erklärt, die Optionen beschrieben, gegebenenfalls Workflows verlinkt, die den jeweiligen Node nutzen, und die Nodes genannt, die oft auf den beschriebenen Node folgen. Dabei sind die meisten REST-Nodes ausführlicher beschrieben als die MongoDB-Nodes. Die MongoDB-Nodes werden meist nur mit einem Absatz beschrieben und es sind selten Workflows verlinkt, in denen sie genutzt werden.

Sowohl zu REST als auch zu MongoDB gibt es „Related workflows“, die auf der Seite der jeweiligen Erweiterung einsehbar sind. Anhand von diesen soll die Nutzung der Nodes nachvollzogen werden können („KNIME MongoDB Integration“, o. J.; „KNIME REST Client Extension“, o. J.).

5.4.2 Catmandu

Die Distribution Catmandu-OAI-0.19 bietet verschiedene Funktionen, um mit OAI-Schnittstellen zu interagieren. Für OAI gibt es einen *Importer* und einen *Store*. Die Daten können umgewandelt werden. Es können einzelne Datensätze oder Sets herausgezogen werden oder auch nur bestimmte Teile der Daten, etwa die Identifier oder Metadatenformate. Zudem ist es möglich, den Zeitraum einzugrenzen (Hochstenbach, 2019b, 2019a). Neben dieser Anleitung finden sich im Bereich „Cookbook“ der Dokumentation auch noch Hinweise auf die Arbeit mit OAI-PMH („LibreCat/Catmandu“, 2016b).

In Catmandu gibt es zwei Distributionen, die mit der Nutzung von REST-Schnittstellen in Zusammenhang stehen. Dies sind Catmandu-Store-REST-0.01 und Catmandu-FedoraCommons-0.5. Die erste Distribution ermöglicht es, eine REST-Schnittstelle als *Store* in Catmandu zu verwenden. Die Schnittstelle muss dazu JSON als Datenformat verwenden und die URL muss dem Format [base_url]/[id][query_string] folgen. Man kann einzelne Elemente abrufen, hinzufügen, ändern oder löschen (Praetere, 2017).

Die zweite Distribution, die in Zusammenhang mit der REST-Schnittstelle steht, ist Catmandu-FedoraCommons-0.5. Dabei handelt es sich um eine Distribution mit vielen verschiedenen Modulen und einer ausführlichen Beschreibung. Angesprochen werden kann mit dem Modul die REST API von Fedora Commons (Hochstenbach, 2018a). Da sich die Distribution speziell mit der REST-Schnittstelle der Fedora-Software befasst, wird sie hier nicht weiter beschrieben.

Für MongoDB gibt es die Distribution Catmandu-Store-MongoDB-0.0803. Mit dieser können MongoDB-Datenbanken angesprochen werden. Die Datenbank wird dabei als *Store* angegeben, ein einzelnes Segment (im Falle von MongoDB also eine Datei) wird als *Bag* angegeben. Die Datenbank ist durchsuchbar. Es können Elemente hinzugefügt oder gelöscht werden, die Inhalte der Datenbank in ein anderes Format umgewandelt werden, gezählt werden etc. (Steenlant, 2019a).

Die jeweiligen Distributionen haben eine Übersichtsseite, auf der eine kurze Beschreibung des Zwecks und der Optionen sowie diverse Beispiele für mögliche Nutzungsfälle gegeben werden. Auch einige der Module haben eigene Unterseiten, in denen sie näher beschrieben werden.

5.5 Leistungsfähigkeit

Das letzte Kriterium, das betrachtet werden soll, ist die Leistungsfähigkeit des Tools. Damit ist gemeint, inwiefern es geeignet ist, auch mit großen Datenmengen umzugehen. Dies ist wichtig, da die Anzahl an Records in Bibliotheken schnell in die Zehntausende geht. So umfassen etwa die *Testdaten* der Deutschen Nationalbibliothek für die Reihe O (Onlinepublikationen) 36.219 Records (Otest.mrc.xml vom 04.07.2019, https://data.dnb.de/testdat/). Besonders, wenn eine neue Software eingeführt wird und Daten in diese eingespielt werden müssen, ist es nötig, große Mengen an Daten gleichzeitig verarbeiten zu können.

Fakten, anhand derer sich die Leistungsfähigkeit quantifizieren lassen, sind die Anzahl an gleichzeitig verarbeitbaren Datensätzen, die Geschwindigkeit, mit der die Verarbeitung einer bestimmten Anzahl an Datensätzen erfolgt, und die Möglichkeit, die Software auf einem Server laufen zu lassen. Die Anzahl an gleichzeitig verarbeitbaren Datensätzen ist wichtig, um nicht für jeden neuen Abschnitt an Records einen neuen Prozess starten zu müssen oder dieses Prozedere aufwendig zu automatisieren. Die Geschwindigkeit der Verarbeitung ist wichtig, um vergleichen zu können, welches Tool dieselbe Aufgabe in kürzerer Zeit erledigt. Damit soll gewährleistet werden können, dass die Verarbeitung von großen Mengen an Daten nicht derartig lange dauert, dass andere Prozesse der Bibliothek behindert werden. Die Möglichkeit der Anbindung an einen Server ist wichtig, da dort mehr Rechenleistung und damit kürzere Verarbeitungszeiten zu erwarten sind. Die Anzahl an Records und die Geschwindigkeit sind direkt quantifizierbar, die Anbindung an den Server ist eine ja/nein-Frage. Mit diesen drei

Fakten sollte eingeschätzt werden können, welches Programm für welche Mengen geeignet ist.

Jedoch ist es nicht möglich, die Geschwindigkeit und die Höchstanzahl an Records im Rahmen der Arbeit praktisch auszutesten, da dies sowohl Umfang und Rahmen der Arbeit als auch die vorhandenen Vorkenntnisse übersteigt. Damit bleibt nur die Möglichkeit zu recherchieren, ob eine Serveranbindung möglich ist (wie beim ersten Kriterium „Plattformen und Aktualität“ bereits geschehen) und ob auf der Webseite Zahlen genannt sind, wie viele Records circa verarbeitet werden können. Letzteres ist aber vermutlich schwierig, da gerade KNIME für eine Vielfalt an unterschiedlichen Arten von Daten geeignet ist, nicht nur für bibliografische Records.

5.5.1 KNIME

Auf die Frage, welche Menge an Daten mit KNIME verarbeitet werden kann, wird in den Frequently Asked Questions geantwortet, dass es prinzipiell kein Limit gäbe, da die Daten auf eine geeignete Weise zwischengespeichert werden. Dennoch könne es vorkommen, dass manche Prozesse zu lange dauern würden oder zu viel Speicher benötigen würden („FAQ“, o. J.-b). Wie in Kapitel 5.1.1 beschrieben, besteht die Möglichkeit, KNIME auf einem Server zu verwenden.

5.5.2 Catmandu

Bei Catmandu finden sich ebenfalls keine genauen Zahlen, wie viele Records verarbeitet werden können. In der Einführung findet sich nur die Angabe, dass Catmandu auch in größeren Programmierprojekten Anwendung finden würde, wobei es mehrere Millionen Records pro Tag verarbeite (Catmandu, o. J.-a). Auch Catmandu kann, wie in Kapitel 5.1.2 beschrieben, auf einem Server betrieben werden.

6 Analyse anhand der entwickelten Szenarien

6.1 Filtern aller englischsprachigen Ressourcen

Beim ersten Szenario sollen aus einem Datensatz alle englischsprachigen Ressourcen herausgefiltert werden. Dies könnte etwa relevant sein, wenn im OPAC eine Facette zum Filtern nach Sprachen vorzufinden ist. Mit dem Szenario wird aufgezeigt, wie man die Daten aus einem speziellen Feld des MARC-Records ausliest und auf Basis dessen bestimmte Records auswählt. Das Grundprinzip entspricht somit einer klassischen known-item-search. Ein ähnliches Beispiel aus der Literatur ist etwa die „Selektion der psychologierelevanten Dokumente […] über die DDC-Klassifikation, prinzipiell kommt auch die Instituts-ID in Frage“ (Herb, Oberländer, & Leidinger, 2008, S. 553).

Datengrundlage sind Testdaten aus der Reihe A der DNB (Atest.mrc.xml vom 25.06.2019, https://data.dnb.de/testdat/). Für das Filtern nach der Sprache kann Feld 041, der Sprachcode, genutzt werden. In dieser Arbeit wird dabei Unterfeld a verwendet, da dieses sich auf den Haupttext bezieht. Das Ergebnis des Szenarios soll eine Liste aller Titel sein, bei denen in einem der 041a-Felder der Code „eng“ für englisch angegeben ist.

6.1.1 KNIME

Das Problem war lösbar. Es dauerte jedoch 8 Stunden und 30 Minuten, da viel Vorarbeit geleistet werden musste, um die Records korrekt einzulesen und die Felder mittels XPath adressieren zu können.

Zuerst wurde versucht, mittels XPath einzelne Inhalte aus dem XML-Dokument auszulesen. Da die XML-Beispiele in KNIME sehr simpel sind, wurde in „Coding with XML for efficiencies in cataloging and metadata“ (Cole et al., 2018) recherchiert, wie Tags und Codes in XPath angegeben werden können – mit datafield[@tag="245"] beziehungsweise subfield[@code="c"]. Dennoch hat der angegebene XPath keine Daten ausgelesen.

In den Einstellungen des Nodes ist ein Ausschnitt der XML-Daten zu sehen. Es wurde mit dem Cursor ein beliebiges Feld angewählt, daraufhin erschien bei „Selected XPath“ der Pfad zu genau diesem Feld. Dieser war

```
/dns:collection/dns:record[2]/dns:datafield[11]/dns:subfield[4]
```

Im Vergleich zu dem bisher genutzten XPath fiel auf, dass jeweils noch „dns:“ mit angegeben ist. Daher wurde der genutzte Pfad entsprechend angepasst. Um das Titelfeld 245a zu extrahieren ist somit folgender XPath nötig:

```
/dns:collection/dns:record/dns:datafield[@tag="245"]/
dns:subfield[@code="a"]
```

Ziel war es nun, die Daten in KNIME so zu formatieren, dass in jeder Zeile der Tabelle ein Record steht. Damit sollte es möglich werden, jene Records zu entfernen, deren Ressource nicht englischsprachig ist. Wenn – wie bisher – das ganze XML-Dokument in einer Zeile steht, ist es nicht möglich, die Records einzeln anzusprechen. Daher wird versucht, die Daten so zu strukturieren, dass in jeder Zeile ein Record steht.

Daher wurde versucht, mit dem Cell Splitter-Node die XML-Zelle zu spalten. Es konnte angegeben werden, dass nach jedem </record> gespalten werden soll. Damit wurde jeder Abschnitt in eine eigene Zelle geschrieben, jedoch wurde für jeden Record eine neue Spalte angelegt. Ziel war aber, jeden Record in eine Zeile zu bekommen. Daher wurde versucht, mit dem Node Transpose die Spalten in Zeilen umzuwandeln, was an sich auch funktionierte. Aber danach kann kein XPath mehr auf die Spalte mit den Records angewendet werden, da die Spalte nicht mehr XML heißt. Daher wurde die Spalte in XML umbenannt – dann erscheint aber (wie inzwischen erwartet) eine Fehlermeldung, dass die Spalte XML kein XML enthält und daher kein XPath angewendet werden kann. Dieser Weg hat also nicht funktioniert.

Nach Recherchen zu weiteren Nodes wie Table Column to Variable und Ungroup, die auch nicht zum gewünschten Ergebnis führten, wurde versucht, XPath zu nutzen. So wurde versucht, sowohl ID als auch die Sprache auszulesen und diese zu matchen. Per Zufall wurde nochmal auf die XML Reader-Konfiguration geklickt, dabei fiel der Punkt „XPath Filter“ auf. Dieses wurde bisher nicht weiter beachtet, denn die Funktion war nicht ganz klar. Außerdem stand dabei, dass XPath nicht vollständig unterstützt werden würde. Daher wurde angenommen, dasselbe Ergebnis (pro Record eine Zeile) könne durch Nutzung des Xpath-Nodes erreicht werden. Daraufhin wurde noch einmal in der Dokumentation nachgesehen. Dort stand „Only nodes of the document, which match this XPath query, will be read. Each matching node is read in a single data cell.“ („XML Reader“, o. J.) – was genau der gesuchten Funktion entspricht. Daher wurde in das Feld /dns:collection/dns:record eingegeben, was zum gewünschten Ergebnis führt.

Um XPath weiter zu nutzen, ist eine kleine Anpassung nötig – das /dns:collection ist nicht mehr nötig, da es bereits im XML Reader angegeben wird. Der XPath für den Sprachcode lautet somit

```
/dns:record/dns:datafield[@tag="041"]/dns:subfield[@code="a"]
```

Damit konnte nun mit der Lösung des konkreten Problems begonnen werden. Der XPath-Node wurde so eingestellt, dass bei mehrfach vorkommenden 041a-Feldern mehrere Spalten angelegt werden. Um nach dem Sprachcode „eng“ zu filtern, wurde der Rule-based Row Filter gewählt.

Aus der Liste der Spalten wählt man die gewünschte aus (diese wird dann in dem Feld, in dem die Regel geschrieben wird, direkt in der richtigen Form eingefügt). Anschlie-

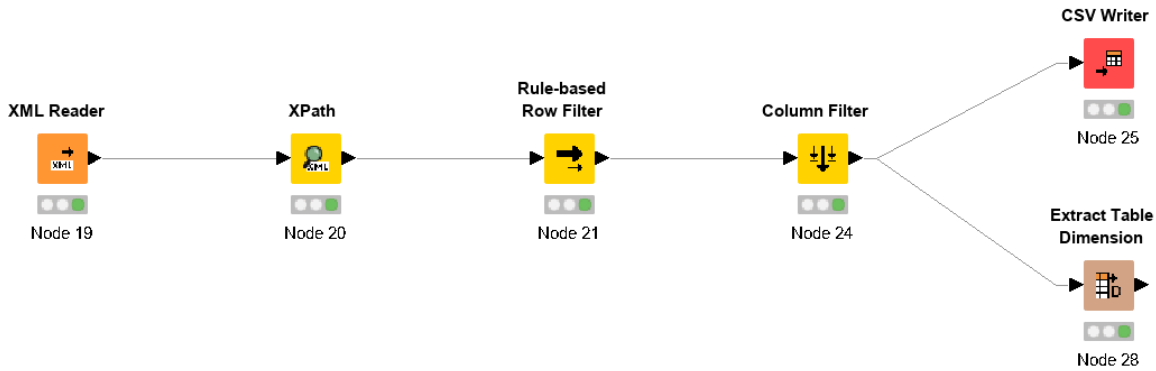
ßend muss man den richtigen Operator suchen. Mit dem Operator = konnte eine funktionierende Regel erstellt werden. Aufgrund der gewählten Option im XPath-Node, bei mehreren Feldern mehrere Spalten anzulegen, gibt es drei Spalten mit Sprachcodes. Da in jeder Spalte nach dem Sprachcode gesucht werden soll lauten die Regeln wie folgt:

```
$language$ = "eng" => TRUE
$language(#0)$ = "eng" => TRUE
$language(#1)$ = "eng" => TRUE
```

An diesen Node konnte ein Column Filter anschlossen werden, mit dem die Spalte mit den Titeln gefiltert wird. Diese kann dann mit dem CSV Reader in eine Datei geschrieben werden.

Zum Schluss sollte noch gezählt werden, wie viele Zeilen die Tabelle hat, damit verglichen werden kann, ob die gleiche Anzahl an Titeln in Catmandu gefunden wurde. Zuerst wurde nach KNIME count rows gesucht und schließlich im Node Repository count eingegeben. Damit wird der Node Value Counter gefunden, der jedoch dazu dient, die Häufigkeit von bestimmten Werten anzugeben. Schließlich wurde aber der Node Extract Table Dimensions entdeckt, der ausgibt, wie viele Zeilen und Spalten eine Tabelle hat. Die Tabelle hat 247 Zeilen.

Somit werden fünf Nodes zum Lösen des Problems benötigt plus einer zum Überprüfen, wie viele Titel gefunden werden. Genutzt wurden die Nodes XML Reader, XPath, Rule-based Row Filter, Column Filter, CSV Writer und zur Überprüfung Extract Table Dimension. Mit den Nodes wird die Datei eingelesen, der Inhalt des Feldes 041a extrahiert, die gesuchten Zeilen herausgefiltert, die Titelspalte separiert und das Ergebnis in eine CSV-Datei geschrieben und gezählt, wie viele Titel gefunden wurden. Zusätzlich zum CSV Writer gibt es noch ein paar andere Exportformate, etwa XLS für Excel und einen Table Writer, aber CSV ist am komfortabelsten für das Setting.



Mit diesem Workflow werden 247 Titel gefunden, bei denen im Feld 041a der Code eng vorkommt.

6.1.2 Catmandu

Das Szenario war in Catmandu innerhalb von 30 Minuten lösbar. Um das Problem herunterzubrechen, wurde erst recherchiert wie ein MARCXML-Feld angesprochen werden kann. Dies scheint mit dem Fix marc_map möglich zu sein. Zusätzlich muss in dem Catmandu-Befehl angegeben werden, um welche Form von MARC es sich handelt, durch die Angabe MARC --type XML. Anschließend wurde nach Beispielen recherchiert. In dem MARC-Tutorial auf metacpan („Catmandu::MARC::Tutorial“, o. J.) wurde ein Beispiel gefunden, in dem alle Records, deren Feld 920a den Inhalt „book“ hat, herausgefiltert werden. Dies entspricht vom Prinzip genau der gestellten Aufgabe. Danke der Kommentare im Beispiel war auch recht verständlich, was zu tun ist und welches Feld wo angegeben werden muss.

In Catmandu sind zwei Schritte nötig, um das Szenario zu lösen: zuerst wird mit einem Texteditor (hier wurde nano verwendet) eine Datei für die Fixes erstellt. Mit dem Befehl „nano filter_language.fix“ wird eine neue Datei mit dem Namen filter_language.fix erstellt und geöffnet. In diese wurden die Fixes aus dem Beispiel kopiert und entsprechend angepasst. Es wurden fünf Fixes verwendet:

```
marc_map(245a,title.$append)
marc_map(041a,language)
select all_match(language,"eng")
select exists(title)
retain(title)
```

Die ersten beiden, marc_map, kopieren den Inhalt des jeweiligen MARC-Feldes in ein zusätzliches Feld. Mit select all_match werden jene Records ausgewählt, auf die das Kriterium (Feld *language* enthält *eng*) zutrifft. Danach werden die Records ausgewählt, die einen Titel haben. Durch retain werden alle Inhalte der Records außer dem Titel verworfen. Damit sollte am Ende eine Liste aller Titel von Ressourcen mit englischsprachigem Inhalt verbleiben. Als zweiter Schritt muss der Fix auf die entsprechende Datei angewendet werden. Dazu kann folgender Befehl genutzt werden:

```
catmandu convert MARC --type XML to YAML --fix filter_language.fix <
Atest.mrc.xml
```

Der Befehlt gibt an, dass eine Datei mit MARCXML-Daten in YAML umgewandelt werden soll. Dabei wird mittels -- fix angegeben, dass die zuvor erstellte Datei filter_language.fix auf die Datei angewendet werden soll. Mit < [Dateiname, hier: Atest.mrc.xml] wird die Datei angegeben, die als Ausgangsdatei dient. So, wie der Befehl hier steht, wird das Ergebnis einfach auf der Kommandozeile ausgegeben. Mit > [Dateiname] kann der Output in eine Datei gespeichert werden oder mit | less kann der Output Seite für Seite angesehen werden.

Um vergleichen zu können, ob die Ergebnisse von KNIME und Catmandu übereinstimmen, sollte gezählt werden, wie viele Titel gefunden wurden. Zuerst wurde nach einer

Funktion in Catmandu recherchiert, die das erledigen kann. Gefunden wurde „count“, womit aber gezählt werden kann, wie häufig ein Titel vorkommt, nicht wie viele Titel es insgesamt gibt. Danach kam die Idee, dass es möglich sein sollte, die Zeilen zu zählen und darüber die Anzahl der Titel zu erfahren. Also wurde recherchiert, wie man in Linux Zeilen zählen kann. Dies geht mit wc -l, wobei wc für das Tool word count steht und die Option -l angibt, dass Zeilen gezählt werden sollen („wc man page“, 2010). Da bei der Ausgabe des Outputs in YAML durch dessen Syntax mehr Zeilen als Titel vorhanden sind, wurde bei dem Catmandu-Kommando auf das Ausgabeformat Text gewechselt. Das Kommando zum Zählen lautet dann:

```
catmandu convert MARC --type XML to Text --fix filter_language.fix < Atest.mrc.xml | wc -l
```

Damit werden 247 Zeilen, also 247 Titel gefunden. Auch die Tabelle in KNIME enthält 247 Zeilen, also 247 Titel. Daher kann angenommen werden, dass beide Programme dieselben Titel gefunden haben, bei denen im Feld 041a der Code eng angegeben ist.

6.2 Filtern aller Ressourcen einer Person aus einem bestimmten Jahr

Beim zweiten Szenario soll nach zwei Kriterien gefiltert werden. Die zwei Felder, die durchsucht werden sollen, sind der erste geistige Schöpfer und das Jahr der Veröffentlichung. Gesucht wird speziell nach allen Werken von 2018, bei denen Wolfgang Pohrt geistiger Schöpfer ist. Die relevanten MARC-Felder sind entsprechend Feld 100a für den geistigen Schöpfer und Feld 264c für das Jahr. Gefiltert wird wiederum die Datei Atest.mrc.xml (vom 25.06.2019, https://data.dnb.de/testdat/). In den Daten sind sechs Ressourcen beschrieben, bei denen Pohrt in Feld 100a erscheint. Eines dieser Werke ist von 2019, die anderen fünf von 2018. Diese fünf sollen gefunden werden, das eine Werk von 2019 hingegen nicht.

Aufbauend auf dem vorherigen Szenario soll gezeigt werden, wie vorzugehen ist, wenn mehr als ein Feld auf die gesuchten Records zutreffen soll. Bei der Suche handelt es sich um eine typische known-item-search.

6.2.1 KNIME

Das Szenario war in KNIME innerhalb von 25 Minuten lösbar. Es gab keine passenden Beispiele, aber das vorangegangene Szenario konnte sehr leicht abgewandelt werden.

Der XML Reader- und der XPath-Node wurden vom vorherigen Szenario übernommen. Beim Rule-based Row Filter wurden die Filterregeln angepasst. Zuerst wurden zwei separate Regeln erstellt, eine für den Autor und eine für das Jahr. Damit wurde allerdings auch das Buch von Pohrt von 2019 gefunden. Das bedeutet aber, dass es aktuell kein Buch gefunden hat, auf das die zweite Regel zum Jahr zutrifft. Wenn diese also

angepasst wird, wird das wohl dazu führen wird, dass alle Bücher gefunden werden, bei denen irgendwie 2018 in 264c steht. Es handelt sich vermutlich um eine logische ODER Verknüpfung und nicht um eine UND Verknüpfung, die eigentlich nötig wäre. Wenn der LIKE-Operator statt des Gleich-Zeichens verwendet wird, erscheinen alle Bücher von 2018 in der Trefferliste. Daher muss ein logisches UND eingeführt werden. Wenn beide Regeln mit UND verknüpft werden, fehlt ein Buch von 2018, da dort nicht exakt 2018 sondern 2018- steht. Die Suche muss also trunkiert werden.

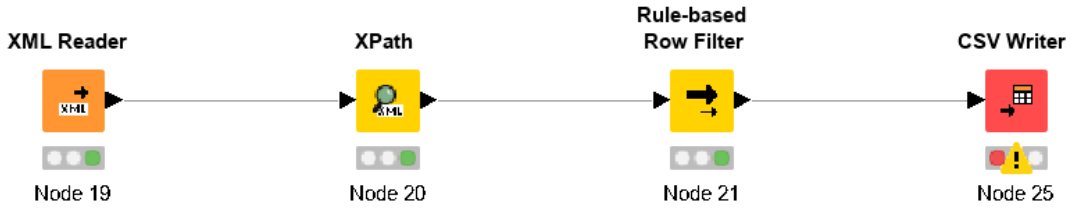
```
($author$ = "Pohrt, Wolfgang") AND ($year$ LIKE  "*2018*") => TRUE
```

Findet alle Bücher von Pohrt von 2018. Mit folgender Änderung:

```
($author$ LIKE  "Pohrt*") AND ($year$ LIKE  "*2018*") => TRUE
```

gelingt auch die Suche ohne die genaue Angabe von Pohrts vollem Namen/dem normierten Sucheinstieg.

Es werden vier Nodes benötigt: der XML Reader, XPath, der Rule-based Row Filter und der CSV Writer.



Es werden die fünf Titel von Wolfgang Pohrt von 2018 gefunden.

6.2.2 Catmandu

Das Problem ist mit dem Tool und dem bisherigen Kenntnisstand lösbar. Benötigt wurden dafür circa 30 Minuten. Das Szenario entspricht dem vorherigen, mit der Ausnahme, dass nun zweimal gefiltert werden muss. Womöglich kann select all_match() einfach zweimal hintereinander geschrieben werden, um nach beiden Kriterien zu filtern. Die Frage ist, ob zuerst auf alle Records die erste Bedingung und dann die zweite Bedingung angewendet wird, oder ob beide gleichzeitig angewendet werden. Bei Letzterem wäre noch die Frage, ob es ein boolesches UND oder ein ODER ist.

Da kein Beispiel mit zwei Kriterien, nach denen gefiltert wird, gefunden wurde, konnte auch einfach getestet werden, ob zweimal select all_match() verwendet werden kann. Die zweimalige Verwendung führt zum richtigen Ergebnis. Daher war die Lösung auch ohne konkretes Beispiel leicht aus dem letzten Szenario abzuleiten.

Zwei Schritte werden benötigt, um das Problem zu lösen: das Schreiben des Fixes und das Ausführen des Befehls. Mit nano filter_pohrt_2018.fix wird ein Dokument mit folgenden Fixes erstellt:

```
marc_map(245a,title.$append)
```

```
marc_map(100a,author)
marc_map(264c,year)
select all_match(author,"Pohrt, Wolfgang")
select all_match(year,"2018")
select exists(title)
retain(title,author,year)
```

Mit marc_map werden die benötigten Felder kopiert. Danach wird mit select all_match gefiltert. Mit retain wird angegeben, dass nur Titel, Autor und Jahr ausgegeben werden sollen. Mit folgendem Kommando können die Fixes angewendet werden:

```
catmandu convert MARC --type XML to YAML --fix filter_pohrt_2018.fix <
Atest.mrc.xml
```

Das Ergebnis stimmt auch mit dem überein, was ich per Suchfunktion im XML Dokument finde – die fünf Titel von 2018, bei denen Pohrt in 100a steht. Die Ergebnisse sind bei KNIME und Catmandu gleich.

6.3 Analyse: Einträge ohne ID finden

Beim einfachen Szenario in der Kategorie Analyse soll überprüft werden, ob in einem gegebenen MARCXML-Dokument jeder Record eine ID hat. Um das festzustellen, wird überprüft, ob das Feld 001 vorhanden ist. Unter der Annahme, dass im bisher verwendeten Datensatz Atest.mrc.xml der DNB jeder Record eine ID hat, wurden in dem Dokument bei fünf Records die IDs entfernt. Mit diesem neuen Dokument soll das Szenario getestet werden. Ausgegeben werden soll zum Schluss eine Liste aller Records, bei denen das Feld 001 nicht vorhanden ist. Zu erwarten ist, dass genau die fünf Fälle gefunden werden, bei denen vorher die ID entfernt wurde.

Fehlende IDs in Datenlieferungen sind ein Problem, dass vom BSZ beschrieben wurde. Wenn bei E-Book-Datenlieferungen Updates kommen, müssen die Records den bereits vorhandenen Datensätzen zugeordnet werden, dazu wird die ID verwendet[1] . Die Überprüfung, ob die ID vorhanden ist, kann in derselben Weise auf andere Felder übetragen werden. Somit wird mithilfe des Szenarios gezeigt, wie man überprüfen kann, ob alle Pflichtfelder belegt sind. Dies kann auch beim Umzug zu einem neuen Programm relevant sein, wenn dort ein Feld als Pflichtfeld hinterlegt ist, welches vorher nicht zwingend notwendig war. Zudem geben Chen et al. (2011, S. 141) an, dass bei der Überprüfung der Qualität von Metadaten untersucht werden soll, ob relevante Felder (beispielsweise Titel, Identifier und Format) belegt sind. Auch Pfeffer (2016, S. 6) beschreibt die Überprüfung auf Vollständigkeit der Datensätze als typische Aufgabe im Metadatenmanagement. In einigen Fällen ist es also nötig zu überprüfen, ob alle gewünschten Felder belegt sind.

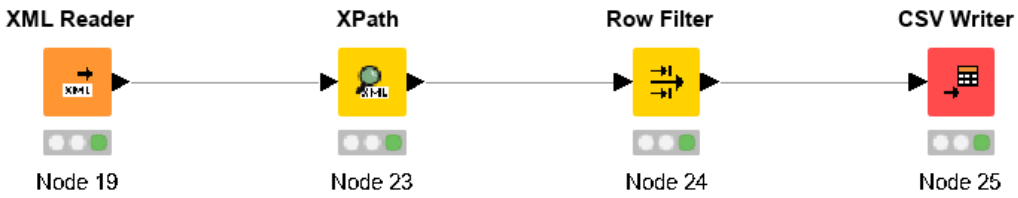
1 Telefonat mit Roswitha Kühn am 16.09.2019, s. Anhang A.1

6.3.1 KNIME

Das beschriebene Problem konnte in KNIME innerhalb von 20 Minuten gelöst werden. Zuerst wurde mit XPath eine neue Spalte angelegt, in welche die ID geschrieben wurde. Diese sollte genutzt werden, um nach fehlenden Werten zu suchen. Da bekannt war, dass es in KNIME einen Node namens „Missing Value“ gibt, wurde zuerst nach diesem recherchiert. Die Suche in KNIME ergab drei Nodes: Missing Value, Missing Value Column Filter und Extract Missing Value Cause.

Die Nodes Missing Value Column Filter und Extract Missing Value Cause konnten nach dem Lesen der Dokumentation ausgeschlossen werden, da Ersteres ganze Spalten löscht, wenn ein gewisser Anteil von Missing Values vorhanden ist, und Letzteres sich mit den Fehlermeldungen der Missing Values beschäftigt. Der Node Missing Value schien nach dem Lesen der Dokumentation und der Betrachtung der Konfigurationsmöglichkeiten nur geeignet, fehlende Werte zu ersetzen, aber nicht, ausschließlich Zeilen herauszufiltern, in denen ein Wert fehlt. Allerdings war in der Dokumentation zu dem Missing Value Node unter den Outgoing Nodes der Row Filter angegeben, sodass dieser Node in Betrachtung gezogen wurde. Im Menü des Row Filters gibt es die gesuchte Option „only missing values match“, was zum gewünschten Ergebnis führt.

Zur Lösung des Szenarios sind damit vier Nodes nötig: der XML Reader, der die Datei einliest, der XPath Node, mit dem die IDs in eine eigene Spalte kopiert werden, der Row Filter, mit dem alle Zeilen, bei denen in der Spalte ID kein Wert steht, herausgefiltert werden, und ein CSV Writer, mit dem das Ergebnis in eine Datei geschrieben wird.



Der Lösungsweg wurde schlussendlich gefunden durch eine Kombination aus dem Hinweis auf den Row Filter in der Dokumentation und dem Gedanken, dass prinzipiell einzelne Zeilen herausgefiltert werden sollen. Mit dieser Grundüberlegung ist das Problem einfach zu lösen. Sucht man jedoch einen speziellen Node für den Umgang mit fehlenden Werten, kommt man nicht direkt auf die Lösung.

Nach passenden Beispielen wurde im Vorhinein nicht gesucht. Es gibt zwar ein Beispiel, in dem auch das Filtern nach fehlenden Werten vorkommt, es heißt jedoch Row_Filtering, sodass man bereits die Herangehensweise oder den passenden Node kennen muss, um es zu finden.

6.3.2 Catmandu

Zur Lösung des Problems in Catmandu wurden 75 Minuten benötigt. Bei den vorherigen Szenarien wurde „select all_match(…)“ verwendet, um diejenigen Records auszuwählen, auf die ein bestimmtes Kriterium zutrifft. Um fehlende Felder zu identifizieren, wird vermutlich ebenfalls select und ein weiterer, anderer Fix benötigt. Durch die Dokumentation auf metacpan (Steenlant, 2019b) wurde der Fix „exists“ gefunden, mittels dem festgestellt werden kann, ob ein Pfad existiert. Da das Feld 001 in einigen der Records in Atest.mrc.xml nicht existiert, da die ID und das Feld gelöscht wurden, sollte dieser Fix verwendet werden können.

Beispiele zur Verwendung von „exists“ finden sich an zwei Stellen: auf der LibreCat-Webseite steht einmal die Schreibweise „reject exits(error.field)“ [fehlendes *s* aus der Dokumentation übertragen] und danach die Schreibweise

```
if exists(error)
   reject()
end
```

Bei beiden ist allerdings die Syntax, in die der Fix eingebettet werden muss, nicht mit angegeben („Catmandu - a data toolkit“, o. J.). Die andere Stelle, an der man Hinweise zur Verwendung von „exists“ findet, ist die Dokumentation auf metacpan (Steenlant, 2019b):

```
# uppercase the value of field 'foo' if the field 'oogly' exists
if exists(oogly)
  upcase(foo) # foo => 'BAR'
end
# inverted
unless exists(oogly)
  upcase(foo) # foo => 'bar'
end
```

Auch hier ist die umgebende Syntax nicht mit angegeben. Zu dem hier angegebenen unless findet sich auf der LibreCat-Webseite („Catmandu - a data toolkit“, o. J.) die Angabe, dass es verwendet werden kann, um Fixes anzugeben, wenn das gesuchte Feld **nicht** existiert. Damit sollten sowohl die Schreibweise „reject exists(…)“ als auch „select unless exists(…)“ genutzt werden können.

Da Unklarheit herrschte, ob das MARC-Feld 001 direkt adressiert werden kann, wurde zuerst folgender Befehl getestet:

```
catmandu convert MARC --type XML to YAML --fix 'select unless
exists(001)' < Atest_fehlende_IDs.mrc.xml
```

Dies führte jedoch zur Ausgabe einer Fehlermeldung („Syntax error […] expected end of input […]“), sodass die Variante mit „reject exists“ getestet wurde. Damit scheinen je-

doch alle Records ausgegeben zu werden und nicht nur die, bei denen Feld 001 fehlt, somit ist die Angabe von 001 als Pfad zum Feld nicht korrekt.

Nach weiteren Recherchen wurde in einer anderen Stelle der Dokumentation („5.1.2 MARC, MAB, PICA paths“, o. J.) der Hinweis gefunden, dass die Angabe von MARC-Feldern wie 001 nur dann funktioniert, wenn man spezielle MARC-Fixes verwendet und dass man bei regulären Fixes nicht einfach MARC-Felder angegeben kann. Daher funktioniert exists(001) nicht, marc_map(001,ID) aber schon.

Die Lösung in Catmandu ist die Verwendung von „reject exists(…)“. Da exists kein Fix ist, der mit Verweisen auf MARC-Felder umgehen kann, muss zuerst der Inhalt des MARC-Feldes in ein zusätzliches, temporäres Feld kopiert werden, dies gelingt mit marc_map. Das neu erstellte Feld kann dann mit exists verarbeitet werden.

Mit nano wurde eine Textdatei namens ID_fehlt.fix erstellt, in diese wurden folgende Fixes eingefügt:

```
marc_map("001",ID)
marc_map("245a",title)
reject exists(ID)
retain(ID,title)
```

Die ersten beiden Zeilen kopieren wie beschrieben den Inhalt des jeweiligen MARC-Feldes (001 bzw. 245 Unterfeld a) in ein separates Feld. Mit der dritten Zeile werden alle Records übersprungen, bei denen eine ID vorhanden ist, das heißt bei denen das Feld 001 vorhanden ist. Zum Schluss werden alle Felder aus den übrig gebliebenen Records außer Titel und (nicht vorhandene) ID entfernt und nur diese Felder ausgegeben. Der Befehl zur Ausführung lautet wie folgt:

```
catmandu convert MARC --type XML to YAML --fix ID_fehlt.fix <
Atest_fehlende_IDs.mrc.xml
```

Das führt dazu, dass die Ausgabe der Records in YAML erfolgt, während die Fixes angewendet werden. Die IDs werden dabei nicht angegeben, da sie ja nicht vorhanden sind. Damit sollte nur nochmal überprüft werden, ob auch tatsächlich bei allen angegebenen Records keine ID verfügbar ist.

In KNIME und in Catmandu werden dieselben Titel gefunden, bei denen keine ID vorhanden ist. Beide Programme zeigen die fünf Records an, bei denen vorher die ID gelöscht wurde. Bei der Ausgabe von Catmandu als YAML fehlt das Feld mit der ID jedoch komplett, während die Spalte im CSV, das von KNIME ausgegeben wird, vorhanden aber leer ist.

Möchte man die Variante mit „unless“ verwenden, muss eine andere Schreibweise verwendet werden. In der Datei mit den Fixes kann statt „reject exists(ID)“ auch genauso

```
unless exists(ID)
select()
```

```
end
```

oder

```
if exists(ID)
reject()
end
```

verwendet werden, nur die Schreibweise in einer Zeile scheint bei der Verwendung von „unless“ nicht möglich.

6.4 Analyse: Einträge mit falschen Codes im Datenträgertyp finden

Im folgenden Szenario sollen diejenigen Records gefunden werden, bei denen der Code im Datenträgertyp falsch ist. Der korrekte Code in Feld 338b kann wichtig sein, wenn dieses genutzt wird, um Online-Ressourcen im OPAC zu filtern (Müller, Smolka, & Severin, 2015).

Mithilfe des Szenarios soll gezeigt werden, wie vorgegangen werden kann, wenn überprüft werden soll, ob ein normiertes Feld immer korrekt belegt ist. Es ist wichtig, überprüfen zu können, ob bestimmte normierte Felder korrekt belegt sind, da eventuell eines der Systeme, mit dem die Bibliothek arbeitet, keine falschen Eingaben akzeptiert sondern mit einer Fehlermeldung abbricht, während das Programm, mit dem die Datensätze erstellt werden, die Eingaben nicht überprüft. Daher muss dem Abbruch des Programms, das keine fehlerhaften Eingaben akzeptiert, vorgebeugt werden. Dieses Problem kann auch bei der Umstellung auf ein neues Programm auftreten. Weiter kann das Verfahren von Nutzen sein, wenn eine Bibliothek in einem Feld normierte Angaben macht, die laut Regelwerk nicht (in dieser Form) vorgesehen sind, aber dennoch auf ihre Korrektheit überprüft werden sollen. Zudem kann das Vorgehen nützlich sein, wenn man filtern möchte, welche Datensätze einen bestimmten Inhalt haben (beispielsweise wenn man eine Liste von Autoren hat, deren Titel aus einer MARCXML-Datei gezogen werden sollen).

Pfeffer gibt als Aufgabengebiet des Metadatenmanagements an, dass „konsistente Feldbelegungen“ (Pfeffer, 2016, S. 6) überprüft werden können und auch Chen et al. (2011, S. 141) fordern die Überprüfung, ob die richtigen Daten im richtigen Feld sind. Ein Teilgebiet davon ist die Überprüfung normierter Felder.

Ausgewählt wurde hier Feld 338 Unterfeld b, der Code für den Datenträgertyp in der Form des RDA Carrier Types („Term and Code List for RDA Carrier Types“, 2019). Dazu wird der Inhalt des Feldes mit einer Liste aller erlaubten Codes verglichen.

Dazu wurde wiederum Atest.mrc.xml (vom 25.06.2019, https://data.dnb.de/testdat/) manipuliert. Es gibt nun je einen Record, bei dem ein Leerzeichen vor oder nach dem korrekten Code steht, bei dem der nicht vorhandene Code „aa“ angegeben ist, bei dem

das zweite Zeichen fehlt oder bei dem das zweite Zeichen doppelt eingetippt wurde. Ziel des Szenarios ist es, eine Liste der Titel mit falschen Codes zu erhalten. Unter der Annahme, dass die nicht manipulierten Daten korrekt sind, sollte man also eine Liste mit sieben Titeln und den beschriebenen manipulierten Codes erhalten.

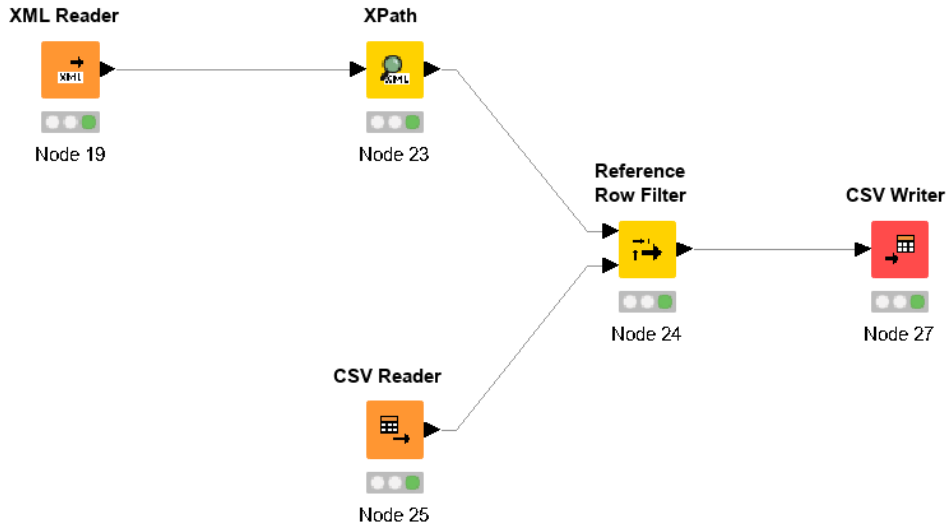
6.4.1 KNIME

Das Szenario war innerhalb von 25 Minuten lösbar. Durch bereits getroffene Vorüberlegungen zum Szenario, bei dem der Campusnetz-Hinweis ergänzt wird (siehe Kapitel 6.6), war klar, dass der Reference Row Filter genutzt werden kann.

Als Beispiel kann hier vielleicht das Advanced_Row_Filtering herangezogen werden. Aber auch hier ist das Problem, dass man das nur findet, wenn man schon weiß, bei welcher Kategorie von Nodes man suchen muss. Fokus der Vorüberlegungen war hier die Arbeit mit einer Liste, mit der die Inhalte eines Feldes abgeglichen werden sollten. Daher war der Name des Nodes, *Reference* Row Filter, ansprechend.

Mit diesem Node können die Inhalte einer Spalte der MARC-Records mit einer Spalte einer CSV-Liste verglichen werden. Daher wurden die Inhalte des Feldes 338b mittels XPath in eine eigene Spalte geladen und mit den zugelassenen Codes aus einer CSV-Datei verglichen. Da teilweise mehrere Codes für eine Ressource vergeben werden, wurde im XPath-Node ausgewählt, dass in diesem Fall „Multiple Rows“ angelegt werden sollen. Das führt zwar dazu, dass für einen Record mitunter mehrere Zeilen angelegt werden, es erleichtert aber das Vergleichen der Codes mit der Liste. Im Falle von mehreren Spalten für mehrere Codes müsste jede Spalte mit der Liste verglichen werden, was den Workflow unnötig vergrößern würde. Wenn man eine „Collection Cell“ anlegen lässt, liegen die Codes in einer Liste vor, was zu Problemen beim Vergleich mit der Liste führen würde. Die Lösung mit mehreren Zeilen führt dazu, dass der Workflow schlank bleibt und auch bei mehreren Codes jeder kontrolliert wird. Sind einer oder mehrere falsch, können die Records über die ID identifiziert werden. Dies funktioniert auch, wenn es einen Record mehrfach in verschiedenen Zeilen gibt. Der Reference Row Filter kann dann alle Records, bei denen der Code korrekt ist, herausfiltern. Dadurch verbleiben nur jene Records in der Liste, bei denen der Code nicht in der CSV-Datei stand.

Es sind fünf Nodes nötig: der XML Reader, der XPath Node, der CSV Reader für die Datei mit den erlaubten Nodes, der Reference Row Filter, mit dem die Inhalte des Feldes mit der CSV-Liste verglichen werden, und der CSV Writer, der das Ergebnis in eine Datei schreibt.



Ausgegeben wird eine CSV-Datei mit ID, Titel und dem falschen Datenträgertyp-Code. Gefunden werden wie erwartet die fünf vorher manipulierten Records.

6.4.2 Catmandu

Das Verfahren in Catmandu ist sehr ähnlich zu dem in KNIME, jedoch dauerte es hier 3 Stunden und 45 Minuten. Dies lag daran, dass Feld 338b in manchen Records mehrfach vergeben ist und erst eine geeignete Datenstruktur gefunden werden musste, mit der die Felder dennoch analysiert werden können.

Es gibt ein Beispiel für genau das Problem, dass man den Inhalt eines Feldes mit einer Liste abgleichen will und alle Records aussortieren will, deren Feldinhalt in der Liste steht („Catmandu::MARC::Tutorial“, o. J.). Im Beispiel werden mehrfach vergebene Felder einfach aneinander gehängt. Dadurch stimmt in diesen Fällen der Inhalt des Feldes nicht mehr mit den korrekten Codes aus der CSV-Datei überein – der Record wird als fehlerhaft ausgegeben.

Der erste Versuch, mit der Datenstruktur umzugehen, war mit marc_has_many die Records auszusortieren, bei denen das Feld mehrfach vergeben ist, um diese gesondert zu behandeln. marc_has_many scheint die Felder aber nicht identifizieren zu können, eventuell weil nicht nur das Unterfeld, sondern das ganze Feld 338 mehrfach vorkommt. Der zweite Versuch war, marc_all_match zu nutzen, da es vielleicht besser mit der Struktur umgehen kann. Aber das geht auch nicht, da der Fix, der in der CSV-Liste nachsieht, ob ein Code existiert, dies nicht direkt mit dem MARC-Feld machen kann.

Danach wurde versucht, die Inhalte des 338b-Feldes nicht als String, sondern als Liste ausgeben zu lassen. Dies geht mit der Option split: 1. Daraufhin ergibt allerdings der Fix, der den Inhalt des Feldes mit einer Liste aller erlaubten Codes vergleicht, einen Fehler. Dies liegt daran, dass nicht die Datenstruktur zur Liste korrekt angegeben ist.

Da in der Dokumentation keine Informationen dazu gefunden wurden, wurde versucht, Arrays so zu adressieren, wie sie in Perl adressiert werden müssen, da Catmandu auf Perl basiert. $carriertype funktioniert allerdings nicht. Danach erfolgt der Versuch, spezifische Elemente des Arrays zu adressieren (mittels index notation). Aber auch zum Beispiel carriertype[0], um das erste Element der Liste zu untersuchen, funktioniert nicht.

Per Zufall wurde in der Catmandu-Dokumentation zum list-Bind ein Beispiel gefunden, das die Datenstruktur von Listen hinreichend erklärte (Steenlant, 2019d). Beim Nachbauen des Beispiels mit den eigenen Fixes konnten die einzelnen Elemente der Liste adressiert werden. In dem Beispiel wird daher über eine Liste iteriert und nur wenn das Element einen bestimmten Wert hat, wird ein Fix angewendet. Aus diesem Beispiel konnte wiederum eine simplere Lösung ohne den List-Bind entwickelt werden, in der jedes Element der Liste mit der CSV-Datei verglichen wird. Dabei half die Sektion zu Wildcards in der Dokumentation. Dort steht, dass man bei Listen die „star notation" benutzt, um alle Elemente einer Liste anzusprechen, also carriertype.* („Catmandu - a data toolkit", o. J.).

Folgende Fixes werden in eine Datei eingegeben:

```
marc_map(338b,carriertype, split: 1)
marc_map(245a,title)
lookup(carriertype.*,'/home/fio/Documents/Uni/Hdm_Stuttgart/Bachelor-
arbeit/Daten/338b_Codes_Datenträgertyp.csv')
reject all_match(carriertype.*,OK)
retain(_id,title,carriertype)
```

Mit folgendem Befehl wird die Analyse durchgeführt:

```
catmandu convert MARC --type XML to YAML --fix check_338b_korrekt.fix
< Atest_falsche_Datenträgertypen.mrc.xml | less
```

Das Ergebnis ist nun bei Catmandu und KNIME gleich.

6.5 Ergänzen von IMD-Typen

Als einfaches Szenario aus der Kategorie „Ergänzen von Inhalten" wurde das Ergänzen von nicht vorhandenen Feldern gewählt. Es handelt sich dabei um ein Beispiel aus der Praxis, das vom BSZ zur Verfügung gestellt wurde. Die Daten sollten für die Universitätsbibliothek Heidelberg in den Fremddatenbestand gespielt werden und stammen von der Bodleian Library. Die Records beschreiben „normale Print-Monos"[2], haben allerdings keine IMD-Typen, daher sollen diese ergänzt werden. Unter den IMD-Typen versteht man die Standardelemente Inhaltstyp, Medientyp, Datenträgertyp. In MARC sind das die Felder 336, 337 und 338. In Unterfeld $a wird jeweils der normierte

2 E-Mail von Gerlind Ladisch an Heidrun Wiesenmüller vom 23.08.2019, s. Anhang A.2

Begriff eingeben, in $b der entsprechende Code und in $2 ein Code für die Quelle des Begriffs und Codes. Die Felder sollten demnach wie folgt belegt sein:

```
336 $aText$btxt$2rdacontent
337 $aohne Hilfsmittel zu benutzen$bn$2rdamedia
338 $aBand$bnc$2rdacarrier
```

(„MARC 21 Format for Bibliographic Data“, 2017; Müller et al., 2015)

Als Datenbasis wurde eine mrk-Datei zur Verfügung gestellt. Da diese weder mit Catmandu noch mit KNIME bearbeitet werden konnte, wurde sie zuerst mit MarcEdit in eine mrc-Datei umgewandelt. Anschließend wurde sie mit Catmandu von mrc zu MARCXML umgewandelt, damit sie auch in KNIME bearbeitet werden kann und im selben Format wie die anderen Dateien, die hier bearbeitet wurden, vorliegt. Die Datei heißt nun nawalkishore_bod_20170227.mrc.xml. Ziel ist es, dass jeder Record mit den entsprechend belegten Feldern 336/337/338 versehen wird.

Anhand des Szenarios soll gezeigt werden, wie vorzugehen ist, wenn man Inhalte in ein noch nicht belegtes Feld einpflegen möchte. Dafür gibt es in der Praxis vielfältige Anwendungsfälle. So schreibt Cole (Cole & Foulonneau, 2007, S. 128), dass nach dem Harvesten von Daten über eine OAI-PMH Schnittstelle immer wieder implizit angenommene Informationen über eine Sammlung dieser explizit hinzugefügt werden müssen. Auch Pfister und Pfeffer nennen das Ergänzen von Feldern als übliche Vorgänge (Pfeffer, 2016, S. 6; Pfister et al., 2017, S. 23).

6.5.1 KNIME

Das Szenario war mit einer Arbeitszeit von 6 Stunden und 25 Minuten lösbar. Grund für die lange Bearbeitungszeit war, dass zuerst zwei erfolglose Versuche unternommen wurden, die nun näher beschrieben werden sollen.

Bei jedem Record – und damit in jeder Zeile in KNIME – sollen die IMD-Typen ergänzt werden. Daher sollte eine neue Spalte angelegt werden, welche die neuen Felder enthält und später mit den ursprünglichen Daten kombiniert werden kann. Zuerst wurde vermutet, dass es nötig sei, eine Schleife zu erstellen, die über jede Zeile iteriert und dabei die Felder in einer neuen Spalte hinzufügt. Es gibt jedoch einen speziellen Node, Constant Value Column, mit dem diese Aufgabe erledigt werden kann.

Danach sollten mit dem XML Column Combiner die Spalte mit dem bisherigen Record und die Spalte mit den neuen Feldern vereint werden. Das Problem dabei ist jedoch, dass zum Verbinden der zwei Spalten ein neues Wurzelelement gesetzt wird, das die beiden Spalten als Unterelemente enthält. Damit ist aber die Struktur des Records nicht mehr gewährleistet. Würde man ein Dokument, dass so erstellt wurde, wieder einlesen, könnte man nicht alle Felder gleich mit XPath ansprechen. Man müsste wissen, in welchem Unterelement sich das gesuchte Element befindet.

Ziel ist es, die Felder so hinzuzufügen, dass sie innerhalb des Elements <record> stehen. Daher sollte geprüft werden, ob das Element entfernt werden kann, um es bei der Vereinigung mit den neuen Feldern wieder hinzuzufügen.

Eine Möglichkeit wäre, den öffnenden und schließenden Tag des Elements record zu entfernen, um es bei der Nutzung des XML Column Combiner-Nodes wieder hinzuzufügen. Insgesamt erschien der Prozess zu aufwendig, als dass es wirklich der richtige Weg sein könnte. Da es aber keine passenden XML-Beispiele gab, wurde es dennoch getestet. Der Start-Tag kann mit dem Cell Splitter-Node entfernt werden. Das Entfernen des End-Tags bereitete mehr Probleme. Nach dem Versuch, reguläre Ausdrücke zu verwenden, gelang es schließlich mit dem String Replacer, den End-Tag zu löschen. Dazu wurde markiert, dass der End-Tag ersetzt werden soll, es wurde jedoch nichts angegeben, mit dem der Tag ersetzt werden soll. So konnte der Tag gelöscht werden. Dennoch konnten die Spalte der bearbeiteten Records und die Spalten mit den neuen Feldern nicht mit dem XML Column Combiner verbunden werden, da dieser nur valides XML akzeptiert, und ohne Wurzelelement ist die Spalte der Records nicht mehr valide.

Eine weitere Möglichkeit wäre, das Dokument beim Einlesen mit dem XML Reader auf einer tieferen Ebene einzulesen. Dadurch würde aber ebenfalls die Struktur der Records verloren gehen. Es wäre nicht mehr erkennbar, welche Felder zu welchem Record gehören, daher ist dies auch keine Lösung.

Als die bisherigen Versuche noch einmal durchgegangen wurden, um nachzuvollziehen, was das Problem war, fiel auf, dass prinzipiell nur ein String eingefügt werden muss. Nodes, die dafür geeignet sind, wurden schon für andere Zwecke in den bisherigen Versuchen genutzt.

Mit dem String Replacer kann der End-Tag des Records, </record>, durch die zu ergänzenden Felder und den End-Tag ergänzt werden. Im String Replacer wurde als Ersatz für den End-Tag folgendes eingegeben:

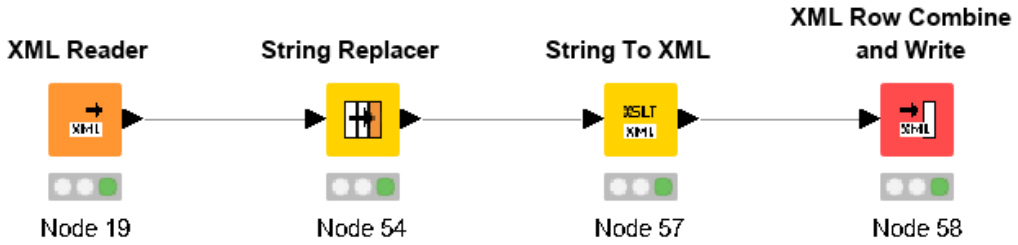
```
<datafield ind1=" " ind2=" " tag="336"><subfield code="a">Text</sub-
field><subfield code="b">txt</subfield><subfield code="2">rdacontent</
subfield></datafield><datafield ind1=" " ind2=" " tag="337"><subfield
code="a">ohne Hilfsmittel zu benutzen</subfield><subfield code="b">n</
subfield><subfield code="2">rdamedia</subfield></datafield><datafield
ind1=" " ind2=" " tag="338"><subfield code="a">Band</subfield><sub-
field code="b">nc</subfield><subfield code="2">rdacarrier</subfield></
datafield></record>
```

Das Ersetzen an sich funktionierte. Jetzt musste allerdings die Spalte wieder in valides XML umgewandelt werden. Dafür konnte der Node String to XML genutzt werden. Der Input wurde auch tatsächlich als valides XML akzeptiert. Schließlich wurde mit dem Node XML Row Combine and Write aus den einzelnen Zeilen wieder ein ganzes XML-Dokument geformt. Dazu musste auch ein neues Wurzelelement angegeben werden.

Dies konnte jedoch <collection> sein, da dies beim Einlesen übersprungen wird und daher sowieso wieder hinzugefügt werden muss. Die XML-Zeichensyntax wird in KNIME automatisch ergänzt. Zu beachten ist, dass der Namespace wieder mit angegeben wird. Dieser wird als Attribut des Wurzelelements collection angegeben. In diesem Fall musste als Name xmlns:marc und als Value http://www.loc.gov/MARC21/slim angegeben werden. Daher muss das Wurzelelement dann auch marc:collection heißen.

Es gibt sicher ein Beispiel für die Anwendung des String Replacers, wenn man allerdings mit XML umgeht und denkt es muss speziell dafür sein, findet man nichts. Dieser Eindruck wurde noch dadurch bestärkt, dass die versuchten Methoden entweder daran scheiterten, dass kein valides XML geformt werden konnte, oder dass bei den vorhandenen Methoden eine zusätzliche Ebene in Form eines weiteren Wurzelelements eingefügt werden musste. Im Endeffekt lag die Lösung dann doch so nah und war sehr simpel.

Bei der Lösung werden vier Nodes verwendet: der XML Reader, der String Replacer, mit dem die Felder eingefügt werden, den Node String To XML, um die Spalte wieder zu XML umzuwandeln (dabei geht es um den hinterlegten Datentyp in KNIME), sowie XML Row Combine and Write, um alle Zeilen zu einem XML-Dokument zusammenzufassen und dieses abzuspeichern. Bei den hinzuzufügenden Feldern muss man auch darauf achten, den Namespace anzugeben.



Zuletzt wurde stichpunktartig kontrolliert, ob überall die IMD-Typen ergänzt wurden. Jetzt ist die Datei wieder einlesbar und sieht genauso aus wie die Ausgangsdatei, nur ergänzt um die IMD-Typen.

6.5.2 Catmandu

Es hat 75 Minuten gedauert, um das Szenario zu lösen. Davon wurden 60 Minuten benötigt, um ein Problem mit den Leerzeichen in einem Fix aufzuspüren und zu lösen.

Es gibt mehrere passende Beispiele für das Problem, die unterschiedlich ausführlich sind und an verschiedenen Stellen der Dokumentation zu finden sind („Catmandu::MARC::Tutorial“, o. J.; Hochstenbach, 2019d). Daher wurde eine Datei mit drei Fixes erstellt, jeweils einmal marc_add für Inhalts-, Medien- und Datenträgertyp. Wird die Datei mit den Fixes jedoch auf den Datensatz angewendet, erscheint eine Fehlermeldung:

```
Oops! Syntax error in your fixes… Error: Expected end of input on line
2 at: marc_add[…]
```

Die Ursache scheint unklar. Ist es nicht möglich, mehrere Felder hintereinander hinzuzufügen? Das wäre ungeschickt. Muss ein Bind verwendet werden, damit die Fixes in einer anderen Reihenfolge abgearbeitet werden? Auch mit do marc_each() lässt sich das Problem nicht lösen. Danach wurde getestet, ob es einen Unterschied macht, wenn man die Fixes nicht in der Datei sondern direkt im Kommando mit angibt. Dabei wurde nicht der gewünschte Text sondern eine sinnlose Zeichenfolge ohne Leerzeichen eingegeben, nur um es zu testen. Das hat funktioniert. Dadurch fiel auf, dass der Unterschied zwischen den funktionierenden Fixes und denen, die eine Fehlermeldung ausgeben, die Leerzeichen sind. Nach einer kurzen Recherche kam die Idee, den Teil mit Leerzeichen in Anführungszeichen zu setzen, was das Problem löst. Leider scheint es in der Dokumentation keine Hinweise zu geben, wann etwas in Anführungszeichen gesetzt werden muss und wann nicht, denn oft ist es nicht nötig, Anführungszeichen zu verwenden.

Nötig ist dreimal der Fix marc_add, mit dem Felder hinzugefügt werden können. Dabei erfolgen getrennt durch ein Komma die Angabe des Feldes, Unterfeldes und dessen Inhalt. Die letzten beiden Elemente können – wenn nötig – wiederholt werden. Falls in dem einzufügenden Feldinhalt Leerzeichen vorkommen, muss der Inhalt in Anführungszeichen stehen.

In eine Textdatei „add_imd_typen“ werden also folgende Fixes geschrieben:

```
marc_add(336,a,Text,b,txt,2,rdacontent)
marc_add(337,a,"ohne Hilfsmittel zu benutzen",b,n,2,rdamedia)
marc_add(338,a,Band,b,nc,2,rdacarrier)
```

Mit folgendem Befehl können den Records die IMD-Typen angehängt werden:

```
catmandu convert MARC --type XML to MARC --type XML --fix add_imd_ty-
pen < nawalkishore_bod_20170227.mrc.xml > nawalkishore_bod_20170227-
mit-IMD-Typen.mrc.xml
```

Dabei wird der Output in eine neue MARCXML-Datei geschrieben.

Bei der neu erstellten MARCXML-Datei wurde stichprobenartig überprüft, ob die IMD-Typen bei jedem Record vorhanden sind. Da dies der Fall war, ist das Ergebnis bei beiden Programmen dasselbe. Nur stehen bei KNIME die Indikatoren vor den Tags in den Datenfeldern. Das scheint beim Einlesen der Datei umgestellt zu werden, sollte aber kein Problem sein.

6.6 Ergänzen des Campusnetz-Hinweises bei E-Books

Als zweites Szenario im Bereich Ergänzen von Inhalten soll den Records, die eine Onlineressource beschreiben, der in Hochschulbibliotheken übliche Hinweis hinzuge-

fügt werden, dass der Zugriff nur aus dem Campusnetz gewährt werden kann. Das Grundprinzip ist das Ergänzen eines Feldes in Abhängigkeit von dem Inhalt eines anderen Feldes. Im Gegensatz zum vorangegangenen Szenario wird hier das neue Feld nur dann angefügt, wenn eine Bedingung erfüllt ist, statt es bei jedem Record anzuhängen.

Neben der Verwendung bei Campusnetz-Hinweisen könnte das Verfahren auch angewendet werden, wenn beispielsweise bei den IMD-Typen oder den Beziehungen zu anderen Entitäten zwar der Code vorhanden ist, nicht aber die entsprechende ausgeschriebene Schreibweise, wie beispielsweise die Beziehungskennzeichnung. Am BSZ wurden zudem vor einiger Zeit[3] die IMD-Typen bei den RAK-Aufnahmen ergänzt, basierend auf zuvor erstellten Regeln, außerdem wurde das frühere Feld 3040 für den Gefeierten umgeändert in 3010 mit entsprechender Beziehungskennzeichnung. Auch dabei wurden Daten in Abhängigkeit von anderen Feldern ergänzt. Im Rahmen der Normalisierung kann es ebenso nötig sein, die Inhalte von Feldern zu analysieren und in Abhängigkeit davon zu ändern (Cole & Foulonneau, 2007, S. 145). Beispielsweise wenn in Dublin Core-Daten ein anderes kontrolliertes Vokabular für den Inhaltstyp verwendet wurde als gewünscht und die Daten daher umgewandelt werden sollen (Cole & Foulonneau, 2007, S. 145). Zudem wird mit diesem Szenario das Verknüpfen der Elemente Filtern und Ergänzen aufgezeigt.

In diesem Szenario wird allen Datensätzen, die eine Onlineressource beschreiben, der Hinweis hinzugefügt. In der Realität geschieht dies natürlich nur bei den Datensätzen, die auch lizenzpflichtig sind, da auch Open Access-Materialien im Katalog sein können. Um das Verfahren zu vereinfachen und das Grundprinzip aufzuzeigen, wird der Hinweis hier auf alle Onlineressourcen angewandt. Entschieden wird das anhand des Feldes 338 Unterfeld b, dem Code für den Datenträgertyp. Wenn dieser „cr“ ist (für Online-Ressource), wird der Hinweis angehängt.

Der Hinweis soll in das Feld 856 Unterfeld z eingefügt werden. Feld 856 (Electronic Location and Access) kann beispielsweise den Link zu der Ressource auf der Verlagswebseite oder einen Persistent Identifier enthalten. Unterfeld z ist vorgesehen für öffentliche Hinweise (public note“. Bei der Recherche nach dem passenden Feld im Katalog des SWB (der eine MARC-Ansicht bietet) wurde festgestellt, dass in den Lokaldaten neben 856z auch 856x, 852x und 852z genutzt werden („Katalogeintrag im SWB mit der PPN 1668655357“, o. J.). x steht dabei jeweils für eine nichtöffentliche Notiz (nonpublic note) und z für eine öffentliche Notiz (public note). Feld 852 (Location) ist gedacht, um die Bibliothek, welche die Ressource besitzt, zu identifizieren. Da der Hinweis für die Nutzerschaft gedacht ist, sollte ein Feld für die öffentliche Notiz (z) gewählt werden. Da in diesem fiktiven Beispiel keine Lokaldaten mehrerer Bibliotheken zusammenkommen, ist dem Feld 856z Vorzug zu gewähren, da dieses explizit für elektronische Medien und deren digitalen Zugriff gedacht ist.

3 E-Mail von Heidrun Wiesenmüller vom 09.08.2019, s. Anhang A.3

Es wurde ein Datensatz aus sieben Records aus den Testdaten der Reihe O (Otest.mrc.xml vom 04.07.2019, https://data.dnb.de/testdat/) und sieben Records aus den Testdaten der Reihe A (Atest.mrc.xml vom 25.06.2019, https://data.dnb.de/testdat/) erstellt. Damit soll gezeigt werden, dass der Hinweis tatsächlich nur bei den Datensätzen angehängt wird, die eine Online-Ressource sind. Am Ende sollen an den sieben Online-Ressourcen der Datei im Feld 856z jeweils der Hinweis „nur aus dem Campusnetz erreichbar / extern via VPN oder Shibboleth“ stehen.

6.6.1 KNIME

Das Szenario konnte innerhalb von 65 Minuten gelöst werden. Der Lösungsweg wurde durch Vorüberlegungen, das Kombinieren der Lösungen der bisherigen Szenarien und der Recherche nach einem geeigneten Node, um die beiden Tabellen zu verbinden, gefunden. Schwieriger war die Suche nach einem geeigneten Muster, anhand dessen das Unterfeld z in das Feld 856 eingefügt werden kann. Da im Gegensatz zum Szenario, bei dem die IMD-Typen ergänzt wurden, nicht ganze Felder ergänzt werden sollen, sondern nur Unterfelder bei schon vorhandenen Feldern, konnte nicht das schließende Element des Records (</record>) genutzt werden. Daher war die Frage, welches Muster im Record nur im Kontext des Feldes 856 auftaucht. Dies ist natürlich der Start-Tag. Da diesesŕ jedoch auch Indikatoren enthält, die je nach Feld unterschiedlich belegt sein können, wurde nach einer Alternative gesucht. Diese könnte der Start-Tag des Unterfeldes u sein, was in diesem Fall auch funktioniert hätte, da das Unterfeld u in den Records nur im Kontext des Feldes 856 genutzt wurde. Bei anderen Unterfeldern wie a wäre dies allerdings nicht möglich, da es bei mehreren, unterschiedlichen Feldern vorkommt. Durch die Dokumentation zu dem String Replacer-Node kam die Nutzung von Wildcards auf. Eventuell könnten dadurch die Indikatoren umgangen werden. Bei dem Versuch, diese Funktion zu nutzen, fällt auf, dass in KNIME die Tags hinter den Indikatoren stehen, auch wenn das in MARCXML eigentlich nicht der Fall ist. Die Struktur ist damit wie folgt:

```
<datafield ind1="4" ind2="2" tag="856">
```

Daher kann in diesem Fall als Muster, nach dem mit dem String Replacer gesucht werden soll, tag=“856“> genutzt werden. Es kann jedoch sein, dass diese Möglichkeit bei anderen Datenstrukturen nicht gegeben ist, da zwischen dem Tag und der schließenden Klammer Attribute angegeben sind, die nicht bei jedem 856-Feld identisch sind. Dann müsste eine andere Option, eventuell mit Wildcards oder regulären Ausdrücken, gewählt werden.

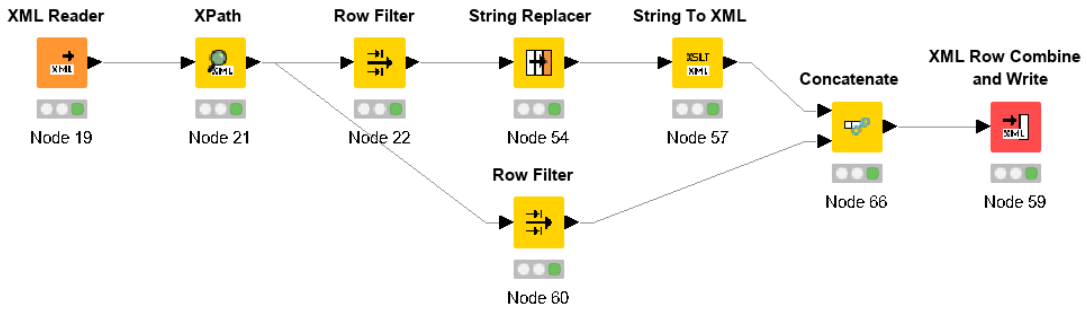
Ein weiteres Problem war das Zusammenführen der veränderten Records mit neuem Feld 856z und den Records, die herausgefiltert wurden, da es sich nicht um Online-Ressourcen handelte. Die beiden Teile sollten wieder zusammengeführt werden, damit alle Records wieder in einer Datei vorliegen und theoretisch weiterverarbeitet werden könnten. Die Auswahl an Funktionen, um zwei Tabellen zusammenzuführen, ist groß.

Vom Namen her wurde zuerst der Joiner benutzt, dieser verknüpft aber die Spalten nicht miteinander, sondern legt die Tabellen nebeneinander. Da jedoch die Zeilen untereinander stehen sollen, wurde bei den Nodes für die Zeilen gesucht und der Node Concatenate gefunden, mit dem wie gewünscht gleichnamige Spalten zusammengeführt werden, sodass die Zeilen untereinander liegen.

Es sind acht Nodes nötig, um das Szenario zu bearbeiten. Zuerst wird mit dem XML Reader die Datei eingelesen. Danach wird mit XPath der Inhalt des Feldes 338b (Code des Datenträgertyps) ausgelesen. Daran schließen sich zwei Row Filter-Nodes an, der erste filtert alle Zeilen, in denen das Feld 338b mit cr belegt ist und der zweite filtert alle, bei denen das Feld 338b *nicht* cr ist. Dies ist nötig, um die herausgefilterten Records später wieder mit den veränderten Records zusammenzuführen. Auf die Records, bei denen 338b mit cr belegt ist, wird ein String Replacer angewandt. Dieser durchsucht alle Zellen der Spalte XML und ersetzt das Ende des Start-Tags von Feld 856 (tag="856">) mit dem selbigen sowie dem zu ergänzenden Unterfeld z. Ersetzt wird das Muster demnach mit folgendem:

```
tag="856"><subfield code="z">nur aus dem Campusnetz erreichbar / ex-
tern via VPN oder Shibboleth</subfield>
```

Dabei wird keine neue Spalte angelegt, sondern die vorhandene abgeändert, damit sie direkt weiterverarbeitet werden kann. Mit dem nächsten Node, String To XML, wird die XML-Spalte dann wieder vom Typ String zum Typ XML umgewandelt. Anschließend wird der Node Concatenate verwendet, um die Tabelle mit den gefilterten und abgeänderten Records der Online-Ressourcen mit den nicht abgeänderten Records zusammenzubringen. Hier wird der zweite Row Filter genutzt, bei dem alle Reihen mit dem Wert „cr“ in der Spalte mit dem Feld 338b herausgefiltert wurden. Der Node Concatenate hängt die Zeilen der zweiten Tabelle an die erste Tabelle an und wenn die Namen der Spalten übereinstimmen, werden diese zusammengeführt. Im Anschluss wird mittels XML Row Combine and Write aus den Zeilen ein einzelnes XML-Dokument mit dem Root-Element „collection“ erstellt und gespeichert. Dieses Dokument kann wieder eingelesen werden und mit XPath durchsucht werden.



6.6.2 Catmandu

In Catmandu konnte das Szenario ebenfalls gelöst werden, es dauerte 1 Stunde 50 Minuten. Zuerst wurde geprüft, wie alle Records ausgewählt werden können, bei denen Feld 338b gleich cr ist. Die Fixes, um Feld 856z anzuhängen, sollten nur auf diese angewendet werden. Die Felder können mit select all_match(Datenträgertyp,“cr“) ausgewählt werden. Daraufhin wird mittels marc_add das Feld 956z mit dem entsprechenden Inhalt angefügt. Dies führt allerdings dazu, dass am Ende jedes Records ein weiteres Feld 856 mit dem Unterfeld z und dem Campusnetz-Hinweis eingefügt wird. Gewünscht ist aber, dass das Unterfeld z in die bestehenden 856-Felder integriert wird.

In der Dokumentation zu MARC findet sich ein Beispiel, in dem gezeigt wird, wie man ein Unterfeld in MARC hinzufügen kann („Catmandu::MARC::Tutorial“, o. J.). Auch wenn die genaue Funktionsweise der Fixes nicht klar erklärt wird, lässt sich das Beispiel auf die ausgewählten Records anwenden und das Unterfeld z wird angehängt. Genutzt wird ein marc_each-Bind, mit dem die folgenden Fixes auf jedes Feld einzeln angewendet werden. Nun sollen die Records, an denen keine Änderungen vorgenommen wurden, wieder hinzugefügt werden. Zuerst wird versucht, mit einem zweiten select() wieder alle Records auszuwählen. Dies funktioniert allerdings nicht, es bleibt bei den zuerst ausgewählten Records. Nach mehreren erfolglosen Versuchen, select() oder reject() zu verwenden, wurde getestet, ob mit dem identity-Bind gearbeitet werden kann. Auch das führte nicht zum gewünschten Ergebnis. Nach einigen Überlegungen kam die Idee, statt dem select bei select all_match() eine if-Bedingung zu verwenden. Damit funktioniert es – die Fixes werden nur auf die Online-Ressourcen angewendet, die restlichen Records bleiben aber dennoch enthalten.

Der Lösungsweg wurde gefunden durch Vorüberlegungen, das Nutzen der bisher erarbeiteten Szenarien, das Beispiel und dann durch Versuchen verschiedener Möglichkeiten (was das Problem mit den nicht angezeigten Records angeht).

Die Schritte zur Lösung sind das Schreiben des Fixes und das Ausführen des Kommandos. Der Fix umfasst neun Zeilen, die zur Lösung nötig sind. Angefügt sind drei weitere Zeilen, die eine Ausgabe von ID, Datenträgertyp und Campusnetz-Hinweis ermöglichen. Die ersten neun Zeilen umfassen eine if-Bedingung, die prüft, ob ein Record eine Online-Ressource beschreibt. Darauf folgt ein marc_each bind, der angibt dass die folgenden Fixes für jedes vorkommende Element ausgeführt werden sollen. Daran schließt sich ein marc_has an, mit dem überprüft wird, ob ein Feld 856 vorliegt. Für jedes Feld 856 wird dann der Campusnetz-Hinweis an eine Variable (hinweis) angehängt, das ursprüngliche 856-Feld gelöscht und durch den Inhalt der Variable ersetzt. Anschließend werden mit „end“ die beiden if-Bedingungen und der Bind geschlossen. Die Fixes lauten:

```
if marc_match(338b,cr)
do marc_each(var:hinweis)
if marc_has(856)
```

```
        add_field(hinweis.subfields.$append.z,"nur aus dem Campusnetz
erreichbar / extern via VPN oder Shibboleth")
        marc_remove(856)
        marc_paste(hinweis)
end
end
end

marc_map(338b,Datenträgertyp)
marc_map(856z,publicnote)
retain(Datenträgertyp,_id,publicnote)
```

Um das Ergebnis ansehen zu können, lautet das Kommando:

```
catmandu convert MARC --type XML to YAML --fix add_campusnetz.fix <
Atest_und_Otest_Auswahl.mrc.xml | less
```

Wenn man die abgeänderten Records in eine neue Datei schreiben möchte, muss man bei dem Fix add_campusnetz.fix die letzte Zeile (retain(Datenträgertyp,_id,publicnote)) durch eine Raute vor dem retain auskommentieren. Auch sollte als Ausgabeformat MARCXML statt YAML verwendet werden. Das Kommando lautet dann wie folgt:

```
catmandu convert MARC --type XML to MARC --type XML --fix add_campus-
netz.fix < Atest_und_Otest_Auswahl.mrc.xml >
Atest_und_Otest_Auswahl_mit_Campusnetz-Hinweis_Catmandu.mrc.xml
```

Hinter der schließenden spitzen Klammer steht der Name der neu zu erstellenden Datei.

Das Feld 856z sieht sowohl im mit KNIME erstellten XML-Dokument als auch im mit Catmandu erstellten XML-Dokument gleich aus. Allerdings steht das Unterfeld z bei XML am Anfang des Feldes 856 und bei Catmandu am Ende. Außerdem fehlt bei KNIME der Namespace „marc:“ vor den datafields und subfields, er ist nur bei „collection“ angegeben. In Catmandu sind die Records in der selben Reihenfolge wie zu Beginn, in KNIME sind erst alle Online-Ressourcen gelistet und danach die restlichen Records. Zudem stehen wie beschrieben die Indikatoren in KNIME vor den Tags.

6.7 Anreicherung der Onlineausgaben mit Feldern der Printausgaben

Beim letzten Szenario soll mit zwei Datensätzen gleichzeitig gearbeitet werden. Hierbei sollen Records der Onlineausgaben mit Feldern der Printausgaben angereicht werden. Dazu wurden aus den Testdaten der Reihen A und O der DNB (aus Atest.mrc.xml vom 25.06.2019 und aus Otest.mrc.xml vom 04.07.2019, https://data.dnb.de/testdat/) jeweils sieben Datensätze ausgewählt. Dabei wurden drei Titel gewählt, die sowohl in den Daten der Reihe A als auch der Reihe O vorkommen. Außerdem wurde jeweils ein

Record gewählt, dessen Titel „Gesammelte Werke“ lautete, die aber von unterschiedlichen Autoren stammten. Damit sollte überprüft werden, ob tatsächlich nur diejenigen Records gematcht werden, bei denen Titel und Autor übereinstimmen. Die restlichen drei Records pro Datensatz waren Titel, die nicht in der jeweils anderen Reihe vorkamen.

Mithilfe dieses Szenarios wird beschrieben, wie vorzugehen ist, wenn man mit zwei Datensätzen parallel arbeitet. Es soll erklärt werden, wie beide eingelesen werden können, einzelne Felder der verschiedenen Datensätze abgeglichen und schließlich Inhalte vom einen in den anderen Datensatz übertragen werden können. In der Praxis kann dieses Beispiel von Relevanz sein, wenn von Verlagen Metadaten für E-Books geliefert werden und beispielsweise aus den bereits katalogisierten Printausgaben die Schlagwörter übernommen werden sollen.

Es wurde festgelegt, dass zwei Records dieselbe Ressource beschreiben, wenn Autor und Titel übereinstimmen. In der Realität müssten weitreichendere Überprüfungen stattfinden, da beispielsweise unterschiedliche Ausgaben des selben Werkes vorliegen könnten. Um die Bearbeitung des Szenarios in der gegebenen Zeit machbar zu halten, erfolgte eine Beschränkung auf die zwei Kriterien Titel und Autor.

Als Feld, das von der Printausgabe in die Onlineausgabe übernommen werden soll, wurde 655 („Index Term-Genre/Form“) gewählt. Daher wurden im XML-Dokument der Onlineausgaben alle 655 Felder entfernt. Da in zwei Records der Printausgaben keine 655-Felder angegeben waren, wurde hier jeweils ein beliebiges Feld eines anderen Records eingefügt. Damit soll gewährleistet werden, dass bei diesen Records ebenfalls ein Feld vorhanden ist, das übernommen werden kann.

Ziel des Szenarios ist es, dass bei den Onlineausgaben, die auch im Datensatz der Printausgaben vorkommen, das Feld 655 ergänzt wird. Der Inhalt des 655-Feldes soll dabei aus dem entsprechenden Record der Printausgabe übernommen werden. Damit sollen am Ende bei drei Records der Onlineausgaben neue 655-Felder enthalten sein.

6.7.1 KNIME

Das Szenario konnte in der vorgegebenen Zeit nicht in KNIME gelöst werden. In KNIME können beide Dateien mit je einem XML Reader eingelesen werden. Anschließend können mit XPath Titel und Autor ausgelesen und in eine eigene Spalte kopiert werden. Nun müssen die beiden Tabellen miteinander in Bezug gesetzt werden. Es soll nachgeschlagen werden, für welche Onlineausgaben auch eine Printausgabe vorhanden ist. Es gibt den Node Reference Row Filter, mit dem die Spalte einer Tabelle mit der Spalte einer anderen Tabelle verglichen und gefiltert werden kann. Allerdings kann dabei immer nur eine Spalte betrachtet werden und nicht die zwei Spalten für Titel und Autor gleichzeitig. Eine Möglichkeit ist, mit dem Reference Row Filter nachzuschlagen, welche Titel der Onlineausgaben auch bei den Printausgaben vorkommen. Ergebnis ist

eine Liste von vier Titeln, die drei Records, die in beiden Datensätzen vorkommen, sowie der Record „Gesammelte Werke“. Dieser muss nun herausgefiltert werden. Ein Versuch war, nun die Autoren der bereits gefilterten Liste mit den Autoren der Printausgaben zu vergleichen, mit einem weiteren Reference Row Filter. Im vorliegenden Fall führt dies zwar tatsächlich zu dem gewünschten Ergebnis, dass nur noch die drei übereinstimmenden Titel enthalten sind, allerdings fiel dabei auf, dass es zu Problemen führen würde, wenn von dem Autor der Gesammelten Werke in der Onlineausgabe ein anderer Titel in den Printausgaben enthalten ist. Ein alternativer Versuch war daher, mit zwei Reference Row Filtern einmal Titel und einmal Autor zu vergleichen. Danach sollten dann diese beiden Ergebnisse miteinander verglichen werden. Theoretisch müsste hier die Datengrundlage angepasst werden, um zu verifizieren, ob dieses Verfahren zu einem korrekten Ergebnis führt.

Das Problem ist nun, dass die 655-Felder kopiert werden müssen. Dabei sollte die XML-Syntax erhalten bleiben. Wird allerdings XPath verwendet, geht diese verloren. Wenn man mit einem XML Reader die Datei einliest und dabei nur die 655-Felder einliest, würde allerdings die Information verloren gehen, zu welchem Record sie gehören. Für dieses Problem wurde daher keine Lösung gefunden.

6.7.2 Catmandu

Das Szenario konnte in der vorgegebenen Zeit nicht in Catmandu gelöst werden. Es konnten keine Beispiele gefunden werden, wie man mit zwei Datensätzen gleichzeitig arbeiten kann. Da über das Kommando mittels < nur eine Datei eingelesen werden kann, muss eine andere Option gefunden werden. Dadurch, dass sowohl Titel als auch Autor miteinander abgeglichen werden müssen und zudem die entsprechenden 655-Felder zwischengespeichert werden müssen, ist das Verfahren sehr komplex.

Prinzipiell wäre es gut, aus den Records der Printausgaben Titel, Autor und die zugehörigen 655-Felder zu extrahieren. Eventuell könnte dafür do hashmap() genutzt werden. Allerdings ist unklar, ob damit die nötige Komplexität erreicht werden kann. Die Idee war, dass dann eventuell mit der Funktion lookup die Records der Onlineausgaben mit der erstellten Datei aus den Printausgaben verglichen werden können. Im Beispiel der Funktion lookup wirkt es aber so, dass nur sehr einfache, flache Datenstrukturen dafür geeignet sind. Deshalb stellt sich die Frage, wie dann die Inhalte der 655-Felder übernommen werden können und vor allem dem korrekten Datensatz zugeordnet werden können. Vielleicht ist eine Möglichkeit die Nutzung der ID, sodass wenn der Titel übereinstimmt, die ID gespeichert wird und wenn der Autor übereinstimmt ebenfalls die ID gespeichert wird. Dann könnte verglichen werden, ob zweimal dieselbe ID vorkommt. Dann müssten noch die 655-Felder über die ID abrufbar gemacht werden. Insgesamt bleibt unklar, wie zwei Datensätze so verknüpft werden können, das man mit beiden arbeiten kann.

7 Fazit

Insgesamt lässt sich sagen, dass Catmandu und KNIME unterschiedliche Vor- und Nachteile haben. Ein großes Problem bei Catmandu ist die Dokumentation. Diese ist auf viele verschiedene Seiten verteilt, wiederholt sich und ist etwas unstrukturiert. Wenn man neu einsteigt, fehlt einem das Wissen, wie die Fixes angewendet werden müssen; also wie die Kommandos um die Fixes herum lauten müssen. Das MARC::Tutorial auf metacpan war sehr hilfreich und auch das Adventskalender-Tutorial eignete sich gut für den Einstieg. Letzteres war am Anfang viel besser verständlich als die allgemeine Dokumentation. Vorteil von Catmandu ist, dass es typische Beispiele aus dem Bibliotheksbereich gibt. Dadurch wird der Einstieg ebenfalls sehr erleichtert.

Die Dokumentation bei KNIME ist besser strukturiert als die von Catmandu und befindet sich gesammelt auf einer Webseite, was ein großer Vorteil ist. Dennoch kann es passieren, dass man etwas nicht findet, was man schon einmal gesehen hatte. Zudem ist die Dokumentation sehr umfangreich, was gerade am Anfang dazu führen kann, dass man sich nicht richtig damit auseinander setzt, da man nicht weiß, was tatsächlich nützlich ist.

Ein Problem bei der Arbeit mit KNIME ist die lange Einarbeitungszeit. Bevor überhaupt mit der konkreten Arbeit an den Daten begonnen werden kann, müssen die Daten korrekt eingelesen und die Nutzung von XPath, um Felder zu adressieren, geübt werden. Beides sind Elemente, die in Catmandu sozusagen von selbst funktionieren, zumindest in den einfachen Fällen. Bei der Arbeit mit Arrays gab es auch in Catmandu Probleme und mangelnde beziehungsweise schwer auffindbare Dokumentation über die Adressierung dieser.

Ein großer Vorteil von Catmandu gegenüber KNIME ist, dass es mit den verschiedenen bibliothekarischen Datenformaten umgehen kann. Zudem ist das Umwandeln von Daten in ein anderes Format sehr einfach. Catmandu ist vermutlich besser geeignet für Aufgaben, bei denen es sich um klassische Aufgaben aus dem Bibliotheksbereich handelt und die schnell erledigt werden müssen, wenn nicht viel Zeit zum Einarbeiten besteht.

Vorteil von KNIME ist, dass es für viele Aufgaben, die häufig im Umgang mit Daten anfallen, eigene Nodes gibt. Insbesondere die Arbeit mit mehreren Datensätzen wird von KNIME sehr viel besser unterstützt als von Catmandu. In der Arbeit mit KNIME war eine wichtige Erkenntnis, dass es für die meisten Sachen einen passenden Node gibt. Wenn ein Weg zu kompliziert erscheint, ist er vermutlich falsch.

Ein weiterer Vorteil von KNIME ist, dass es auf allen drei großen Plattformen und insbesondere auf Windows verfügbar ist, während Catmandu auf Windows nur virtualisiert

werden kann. Viele Bibliotheken arbeiten jedoch mit Windows. Beide werden weiterentwickelt. Bei den Datenformaten und Metadatenstandards ist Catmandu besser aufgestellt und einfacher anzuwenden. Jedoch unterstützt KNIME auch die Arbeit mit XML und RDF. Bei REST-Schnittstellen ist KNIME vermutlich besser unterstützt, da die Catmandu-Version womöglich nicht weiterentwickelt wird.

Bei beiden Programmen ist positiv hervorzuheben, dass sie aktiv weiterentwickelt werden und im Forum beziehungsweise auf der Mailingliste auch von Seiten der Entwickler auf Fragen eingegangen wird.

Bis auf Szenario 7 (Datenanreicherung) konnten alle Szenarien in beiden Programmen zufriedenstellend gelöst werden. Die Bearbeitungszeit war jedoch recht unterschiedlich, so wurden mindestens 20 Minuten benötigt, in einem Fall jedoch 8 Stunden und 30 Minuten. Bei beiden Programmen kam es vor, dass das Problem im Prinzip schnell gelöst war, beziehungsweise der Lösungsweg schnell bekannt war, jedoch viel Zeit für einen spezifischen Teil des Problems aufgewendet werden musste. Die Probleme im Umgang mit den Datenstrukturen unterscheiden sich bei der Arbeit mit KNIME und Catmandu, sind aber bei beiden Programmen eigentlich die Hauptursache für Probleme.

Nach der Einarbeitung in KNIME in die Nodes XML Reader und XPath konnten die darauf folgenden Szenarien größtenteils sehr zügig bearbeitet werden. Ein Ausreißer war noch beim ersten Hinzufügen von MARCXML-Feldern zu verzeichnen. Insgesamt kann bei KNIME gesagt werden, dass es vielfältige Funktionen und Möglichkeiten für die Arbeit mit den Daten liefert; schwierig wird es, wenn mit der bestehenden Struktur der Daten gearbeitet werden muss – etwa beim initialen Einlesen oder bei der Integration neuer Felder in die bestehende Struktur.

Insgesamt entstand der Eindruck, dass XML keines der Formate ist, mit denen in KNIME sehr viel gearbeitet wird. So umfasst die JSON-Erweiterung mehr Nodes und auch in einigen Büchern zu KNIME tauchte XML wenn dann am Rande auf.

Bei Catmandu hingegen ist mit der Komplexität der Szenarien tendenziell auch eine längere Bearbeitungszeit zu verzeichnen. Dafür musste für die einfacheren Aufgaben am Anfang weniger Zeit investiert werden. Viel Zeit wurde bei Catmandu für das Ergänzen der IMD-Typen benötigt. Das Grundprinzip an sich war zwar beschrieben, jedoch scheiterte die Umsetzung an fehlenden Anführungszeichen. Hier wäre es hilfreich gewesen, wenn in der Dokumentation erläutert werden würde, wann Anführungszeichen zwingend nötig sind. Aus den Beispielen in der Dokumentation ist nicht recht ersichtlich, wann diese nötig sind und wann nicht.

Problematisch bei dem Szenario mit der Datenanreicherung war, dass für Catmandu keine Informationen gefunden wurden, wie mehrere Datensätze gleichzeitig verwendet werden können. In KNIME war das recht offensichtlich, dafür bereitet dort die Zuordnung der Felder zu einem bestimmten Record Probleme.

Eine weitere Erkenntnis war, dass es immer wieder sehr hilfreich ist, bei Problemen etwas Abstand zu der Aufgabe zu gewinnen. Wenn man es sich nach einer Weile wieder ansieht, wirkt die Lösung plötzlich offensichtlich. Dies war beispielsweise beim Hinzufügen von Feldern in KNIME der Fall oder als in Catmandu die Fixes nur auf einen Teil der Records angewendet werden sollten.

Wichtig zu wissen bei der Arbeit mit Catmandu war, dass die Notation zum Ansprechen von MARC-Feldern auch nur mit MARC-Fixes funktioniert und daher oft Felder mit marc_map kopiert werden müssen, um sie zu verarbeiten. Auch, dass sie Reihenfolge der Fixes in der Datei wichtig ist, war zentral. Am wichtigsten war jedoch zu lernen, wie die Fixes tatsächlich auf die Daten angewendet werden können, das heißt, wie die Grundstruktur eines Catmandu-Befehls lautet. Dafür waren die Beispiele hilfreich. Dennoch wäre es wünschenswert, wenn es eine Übersicht über die verschiedenen Möglichkeiten der Syntax gäbe, insbesondere mit spezielleren Datenstrukturen. Alternativ könnten auch die Beispiele ausführlicher beschrieben werden. Auch lehrreich war die Verwendung weiterer Kommandozeilen-Tools wie wc oder less.

Wichtigstes Element für die Arbeit mit KNIME war sicherlich die korrekte Adressierung der Felder mittels XPath und das Einlesen der Dateien, sodass jeder Record in einer Zeile steht.

8 Ausblick

Diese Arbeit hat einen kleinen Einblick in die Arbeit mit den Tools Catmandu und KNIME gegeben. Es gibt jedoch noch eine Vielzahl weiterer Tools sowie Untersuchungskriterien und Szenarien, die hier nicht beleuchtet wurden. Auch, weil die Methodik durch die Beschränkung auf eine Testerin begrenzt ist, wären weitere Arbeiten in dem Bereich wünschenswert, insbesondere von Menschen mit anderen Vorerfahrungen.

Die Liste an Softwarepaketen, die für das Metadatenmanagement genutzt werden können, ist lang. Hervorzuheben für weitere Untersuchungen wären metafacture, KriKri, d:swarm und LODRefine (OpenRefine mit einer Linked-Open-Data-Erweiterung). Entsprechende Listen mit relevanten Softwarepaketen finden sich etwa bei FOSS4LIB („Metadata Manipulation“, o. J.), Yang und Li (2015, S. 51ff.), Pfeffer (2016) und in der Catmandu-Dokumentation („Catmandu - a data toolkit“, o. J.).

Insbesondere bei den Datenformaten und Metadatenstandards sowie bei den Schnittstellen und Datenbanken wurde nur ein kleiner Ausschnitt der vorhandenen Möglichkeiten betrachtet. Im Bereich Datenformate und Metadatenstandards wären beispielsweise noch MODS, ONIX oder BIBFRAME zu nennen. Auch die Untersuchung von Standards, die eher im Archiv- oder Museumsbereich anzutreffen sind, wäre möglich. Weitere Schnittstellen, die untersucht werden könnten, sind SOAP, SRU, Z39.50 oder SWORD; bei den Datenbanken wäre sicher auch eine Untersuchung klassischer relationaler Datenbanken wie MariaDB sinnvoll.

Aus den Gesprächen mit dem BSZ ergab sich noch ein weiteres wichtiges Szenario, das in dieser Arbeit nicht betrachtet wurde: falsche Zeichen im Zeichensatz. Das ist insbesondere bei Umlauten oder Sonderzeichen bei Autorennamen ein Problem. Daher ist es nötig, die Lieferungen zu überprüfen und gegebenenfalls fehlerhafte Datensätze herauszuziehen. Auch wäre es sinnvoll, die Nutzung der Schnittstellen und Datenbanken, die hier nur theoretisch untersucht wurde, praktisch auszutesten. Ein Fokus auf den Bereich Semantic Web und Linked Data wäre ebenfalls denkbar, sodass der Umgang mit RDF-Daten und den verschiedenen Serialisierungen untersucht werden könnte.

Anhang A: Kommunikation mit dem BSZ

A.1 Telefonat mit Roswitha Kühn am 16.09.2019 – Notizen

Bibliothekarin, bekommt gemeldet, welche E-Book-Pakete die Bibliotheken haben wollen und wie die Lokaldaten aussehen sollen, und von Verlagen Daten geliefert → gucken, dass Daten über K10plus in SWB kommt und von dort in Lokalsystem → Ebook-Pool

Catmandu wird benutzt um MARCXML in MARC 21 umzuwandeln hauptsächlich, sonst wird viel mit PERL gearbeitet

MARCedit zur Analyse, Ferken - Statistiken von VZG Göttingen selbst entwickelt zur Überprüfung

häufige Probleme und Aufgaben:

- brauchen Daten kodiert in UTF8, sind aber nicht immer korrekt oder falsche Zeichen im Zeichensatz
- 001 fehlt (ID) in MARC-Feld
- Autorennamen wegen Umlauten überprüfen, haben Liste mit Feldern, die überprüft werden
- grobe Grundprüfung – Link enthalten? Identnummer beim Datensatz (um Updates zusammenzuführen), bei der Einspielung wird gematcht und gemergt (Kriterien: ID, URL, wenn mehrere vorhanden gibt es Regeln, welche die richtige ist – wenn nichts gefunden neuer Datensatz, wenn was gefunden wird geprüft und ergänzt).
- Liste der Autoren heraussuchen, auf falsche Zeichen prüfen

Formate:

- viele in MARC21, manche in ONIX
- Daten im CSV-Format akzeptieren sie nicht, da diese meist nicht ordentlich genug gemacht sind
- picaplus ist internes Format, kann auch als MARC ausgegeben werden

Update-Lieferungen werden teilweise ohne Prüfung eingespielt, wenn es gut läuft → manchmal wird daher hinterher analysiert; neue Daten werden aufwändiger bearbeitet

Beispieldatei: typischer Fall – Japan-Datei mit lauter Sonnen drin – alle Datensätze mit Sonnen herausfiltern und löschen, Header abändern damit der wieder stimmt

A.2 E-Mail von Gerlind Ladisch an Heidrun Wiesenmüller vom 23.08.2019

[…]

anbei eine MARC-Datei des Nawal Kishore Bestandes der Bodleyan Library. Die Daten wurden gerade für die UB Heidelberg in den Fremddatenbestand geladen und haben grundsätzlich keine IMD-Typen. Es sind alles ganz normale Print-Monos, Zeichensatz ist Unicode. Teilweise ist Originalschrift enthalten.

Da könnte man beispielsweise MARC 336-338 ergänzen oder auch versuchen in 100 $e (Beziehungskennzeichnung) und $4 MARC-Code zu ergänzen.

[…]

A.3 E-Mail von Heidrun Wiesenmüller vom 09.08.2019

[…]

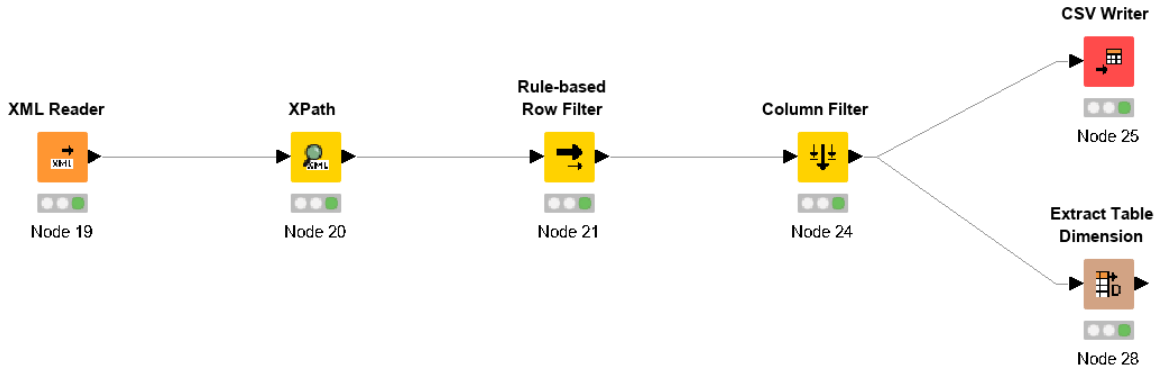
Darüber hinaus macht das BSZ natürlich unendlich viel im Bereich Metadatenmanagement, z.B. immer dann, wenn eine neue Bibliothek in den Verbund kommt. Oder vor einiger Zeit wurden nach bestimmten (zunächst zu erarbeitenden) Regeln die Felder für Medientyp, Datenträgertyp und Inhaltstyp an die alten RAK-Aufnahmen zugespielt oder z.B. das frühere Feld 3040 (speziell für den Gefeierten) umgeändert in ein 3010 mit entsprechender Beziehungskennzeichnung. Und bei der Zusammenführung von SWB und GBV in den K10plus sind natürlich auch unglaublich viele Metadatenmanagement-Aufgaben angefallen (die auch noch nicht alle abgeschlossen sind).

[…]

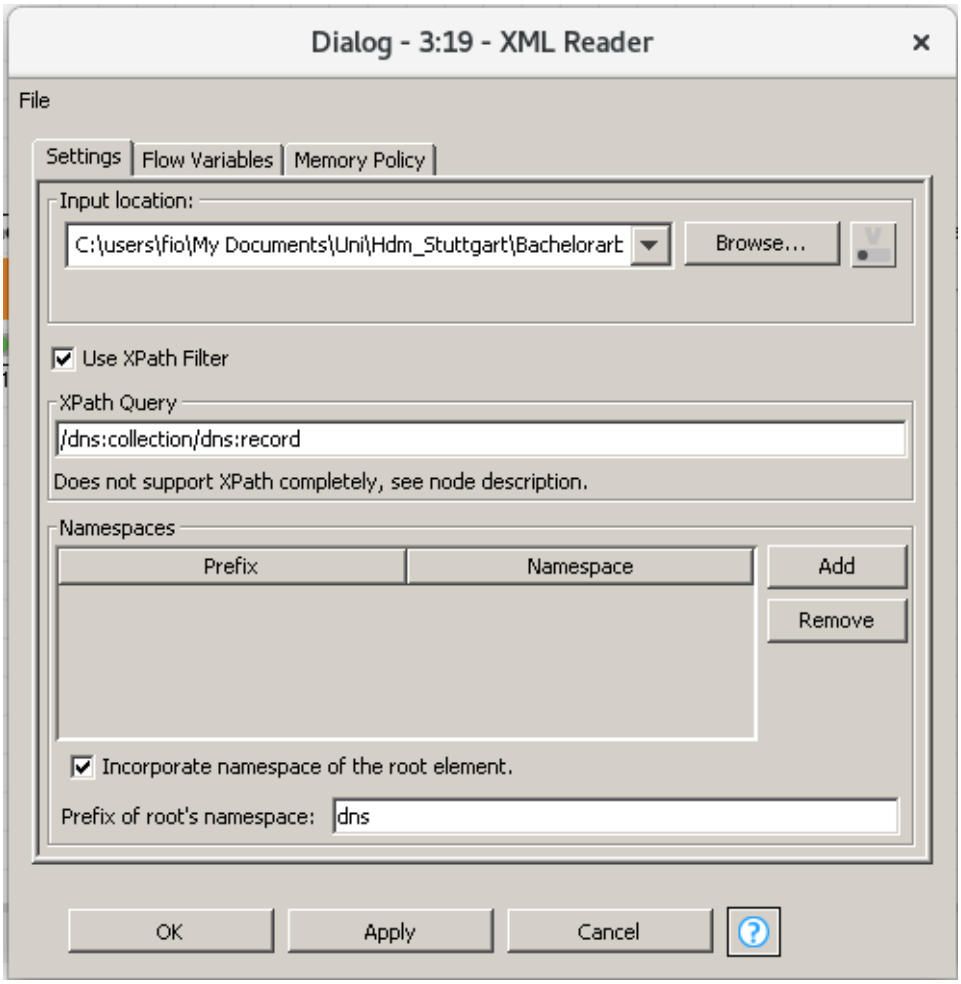
Anhang B: Screenshots der Konfigurationen in KNIME

B.1 Filtern aller englischsprachigen Ressourcen

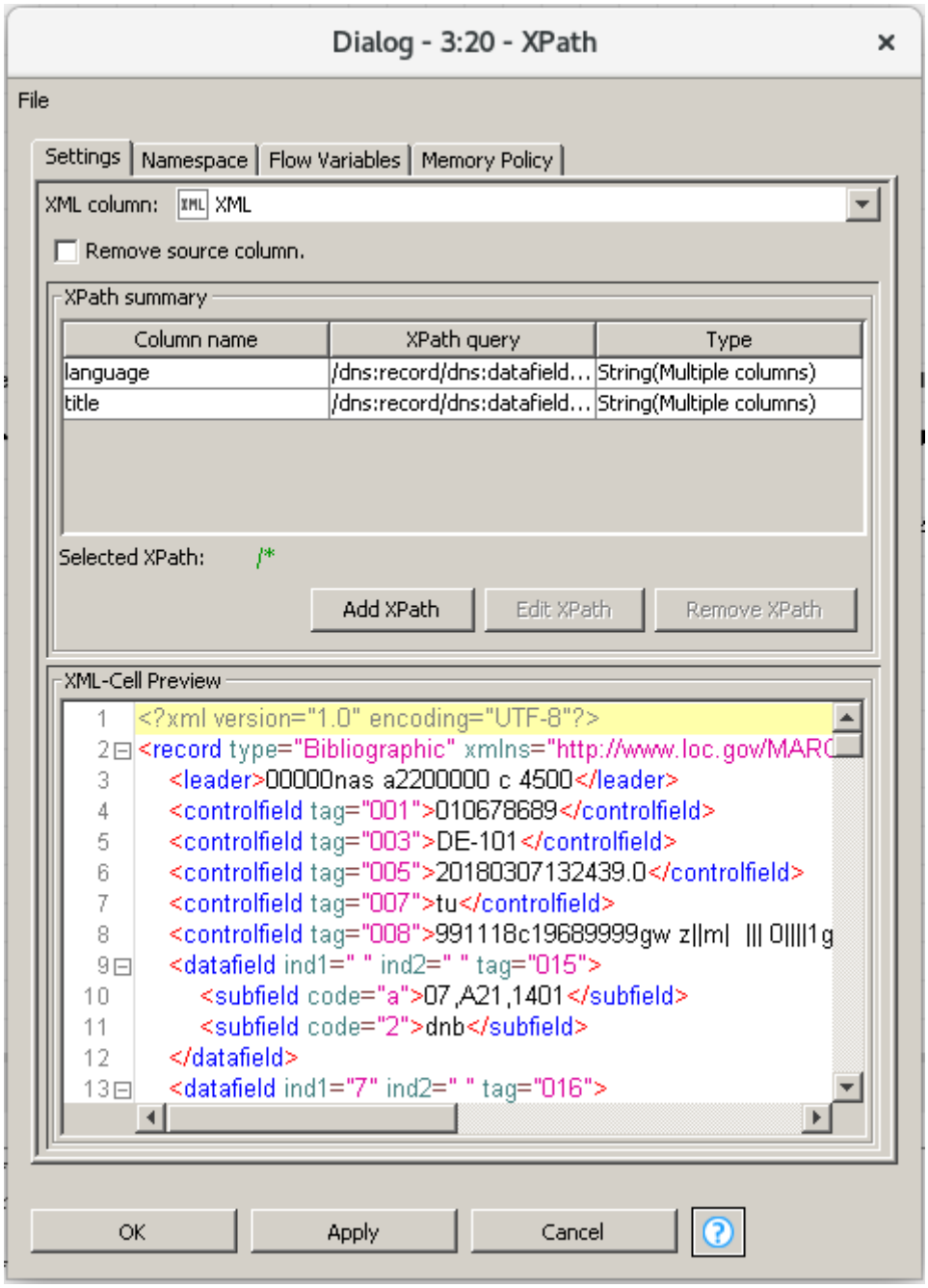
Übersicht:



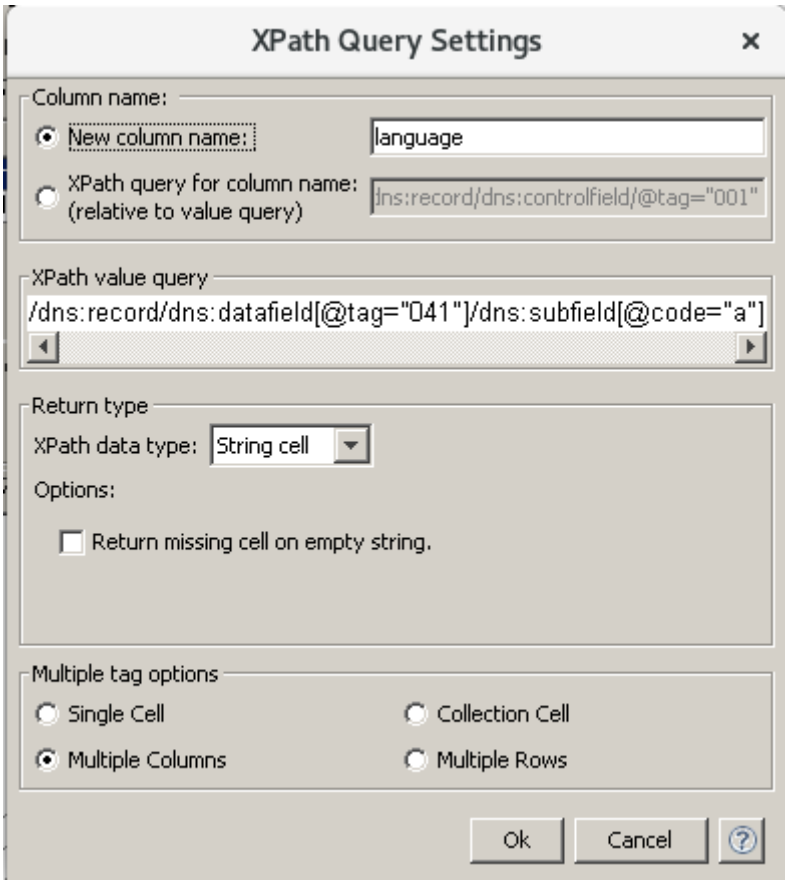
Konfiguration XML Reader:



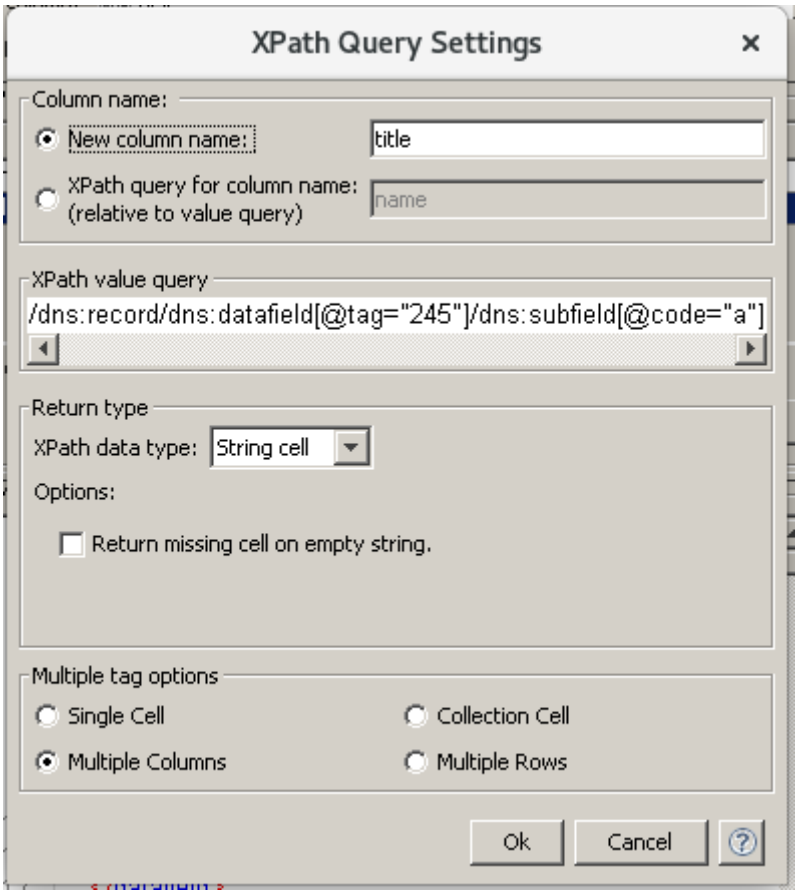
Konfiguration XPath:



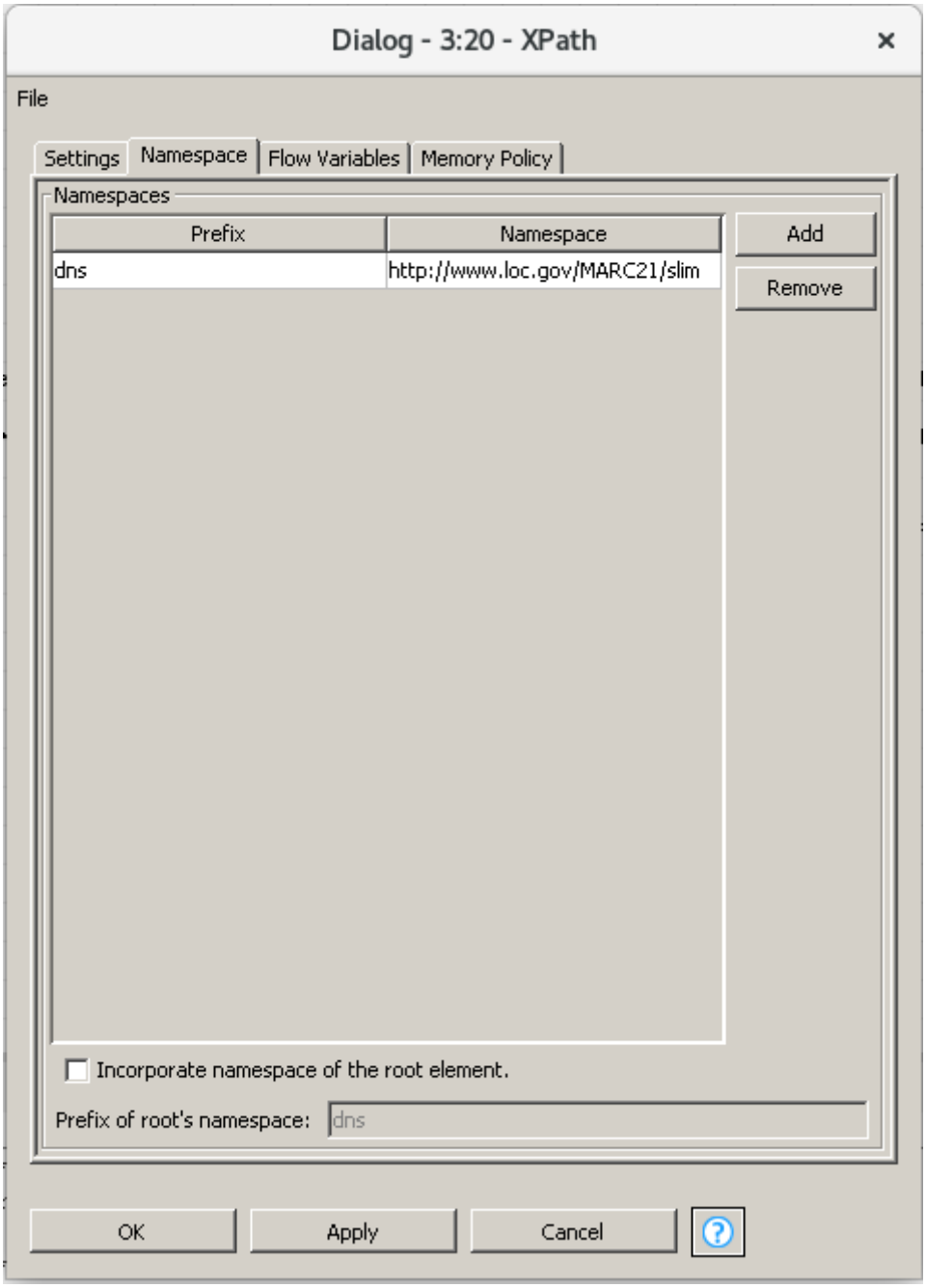
Konfiguration der „language“-Query innerhalb des Xpath-Nodes:



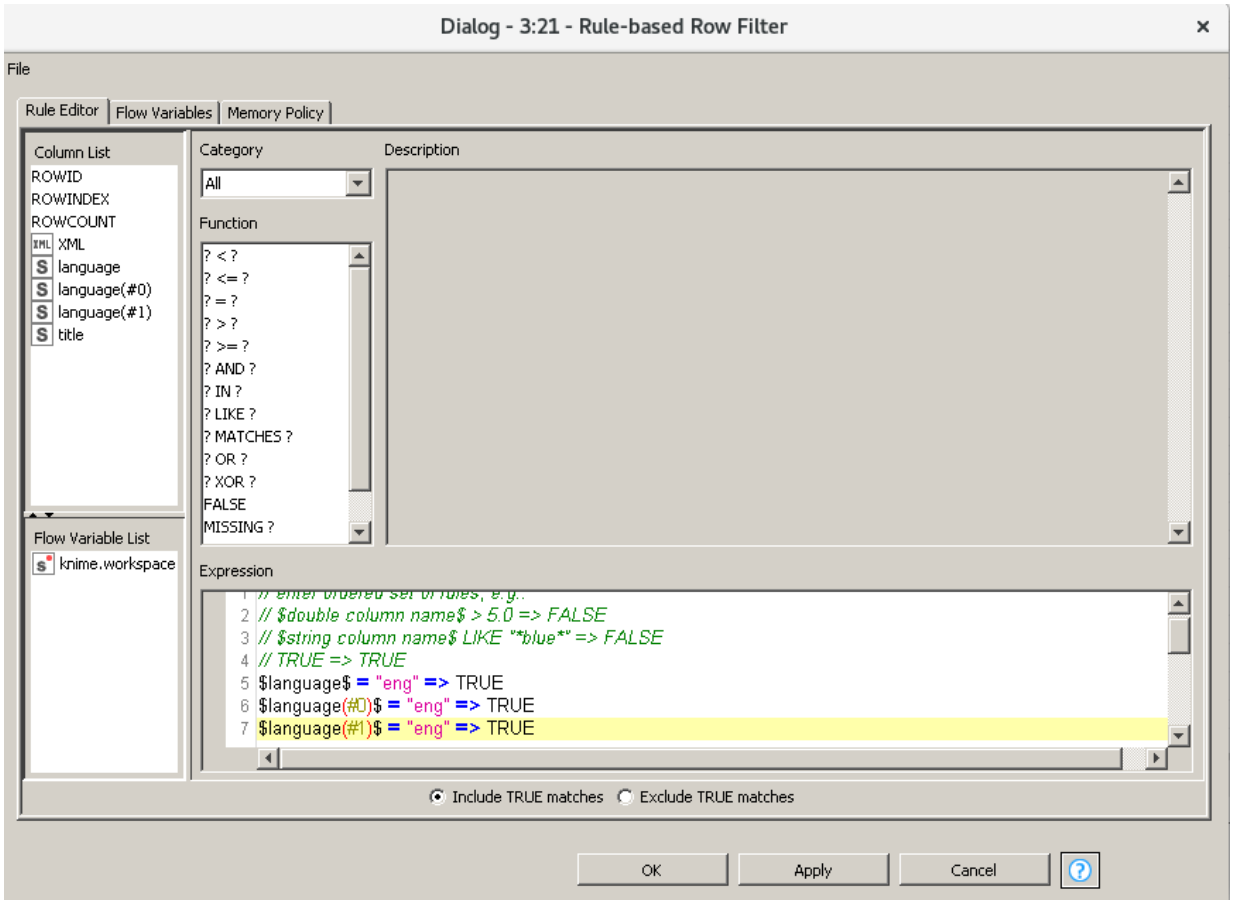
Konfiguration der „title“-Query innerhalb des Xpath-Nodes:



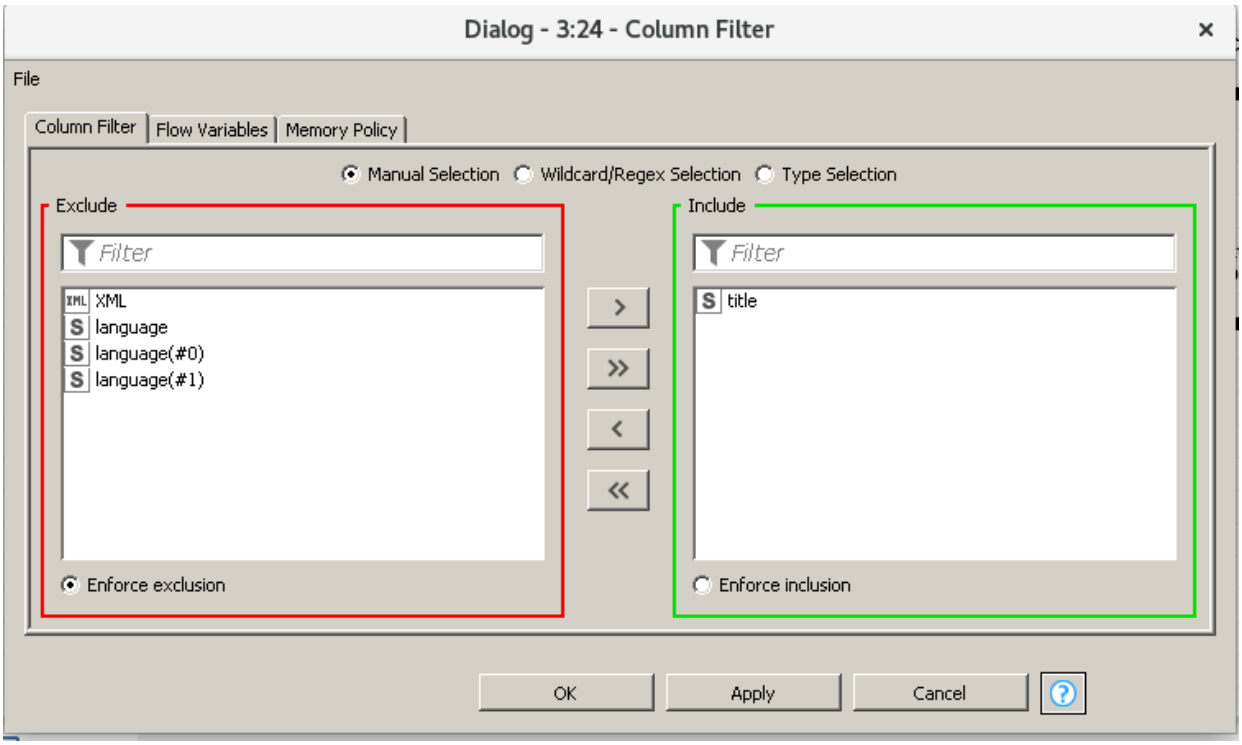
Konfiguration XPath – Reiter „Namespace“ (gegebenenfalls automatisch eingestellt):



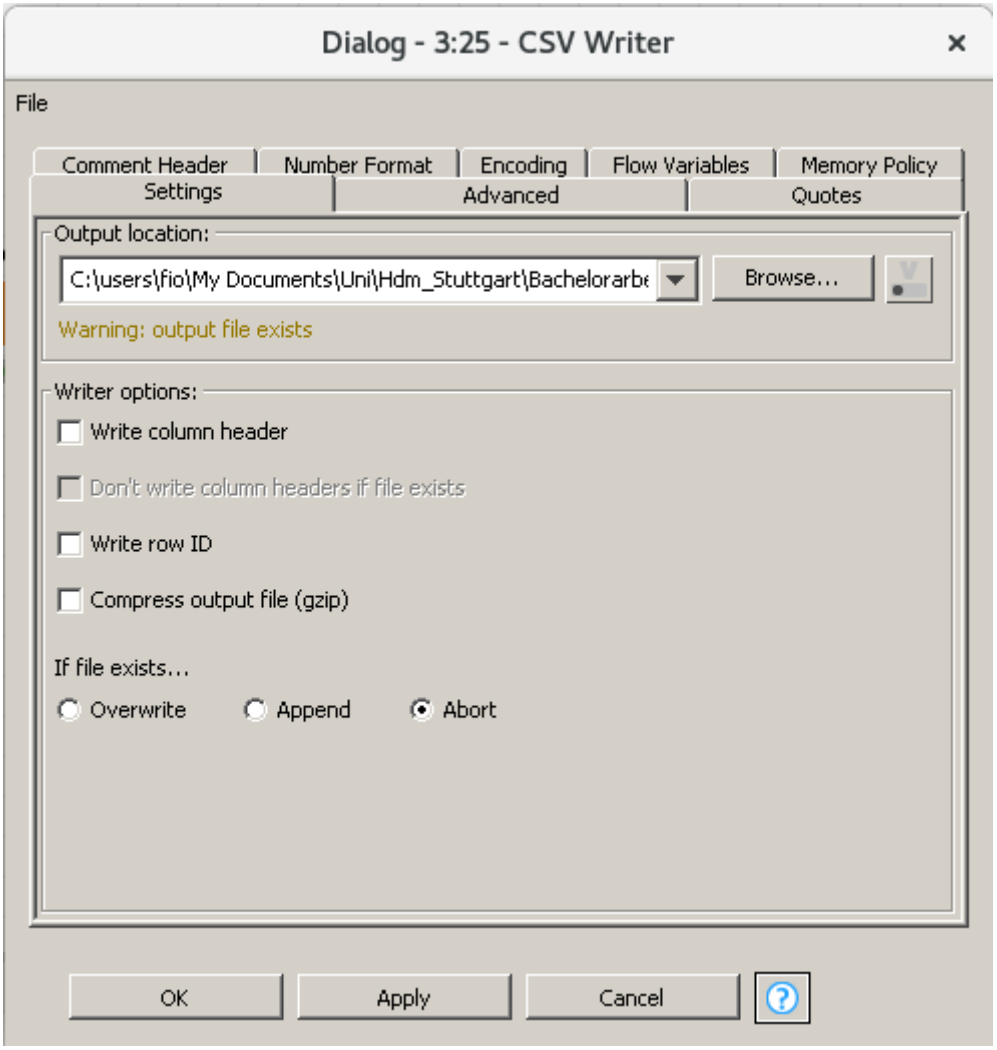
Konfiguration Rule-based Row Filter:



Konfiguration Column Filter:

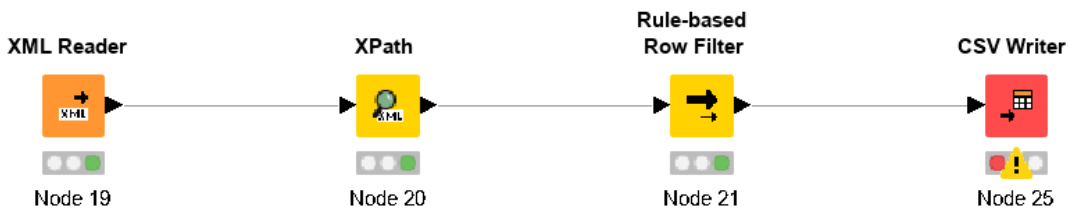


Konfiguration CSV Writer:



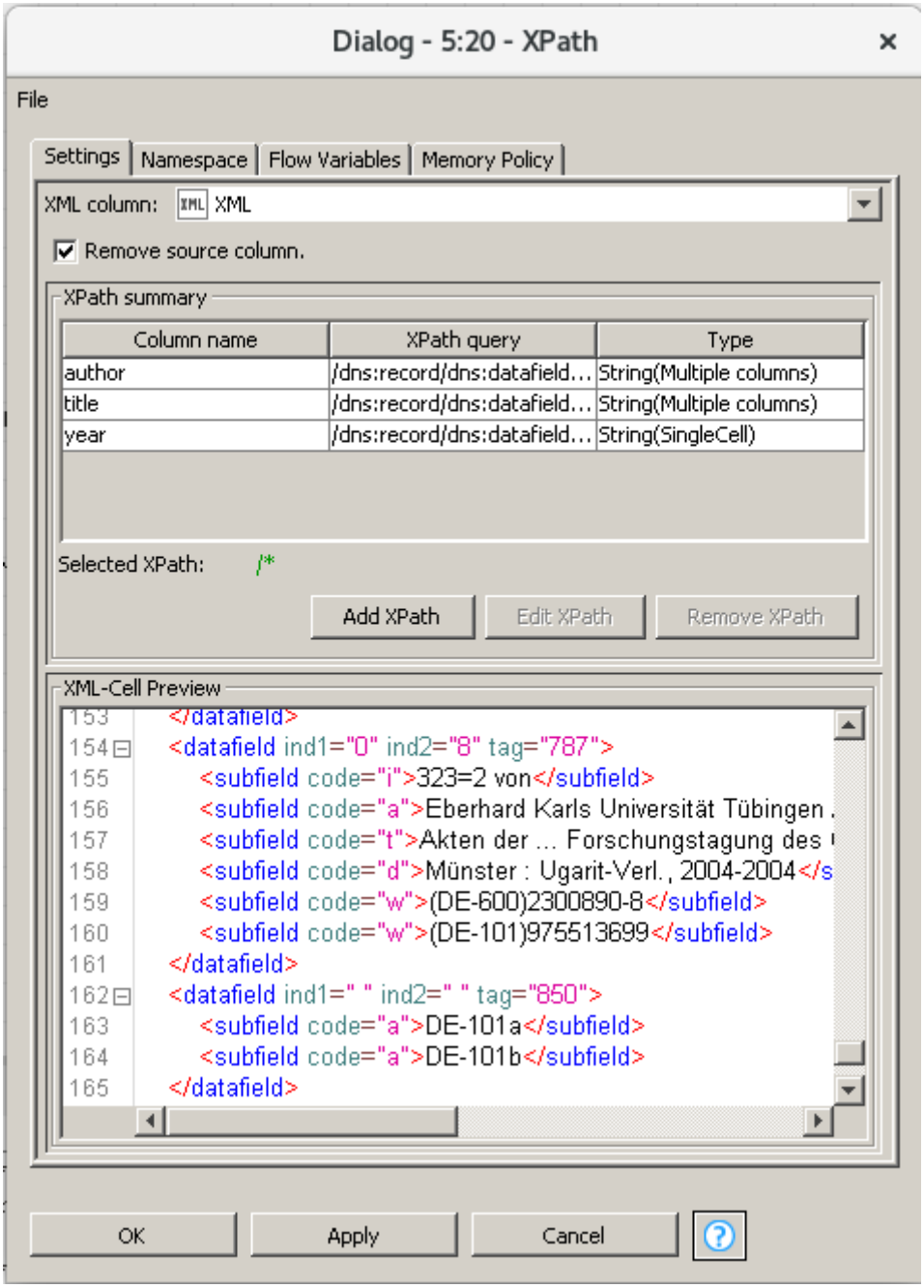
B.2 Filtern aller Ressourcen einer Person aus einem bestimmten Jahr

Übersicht:

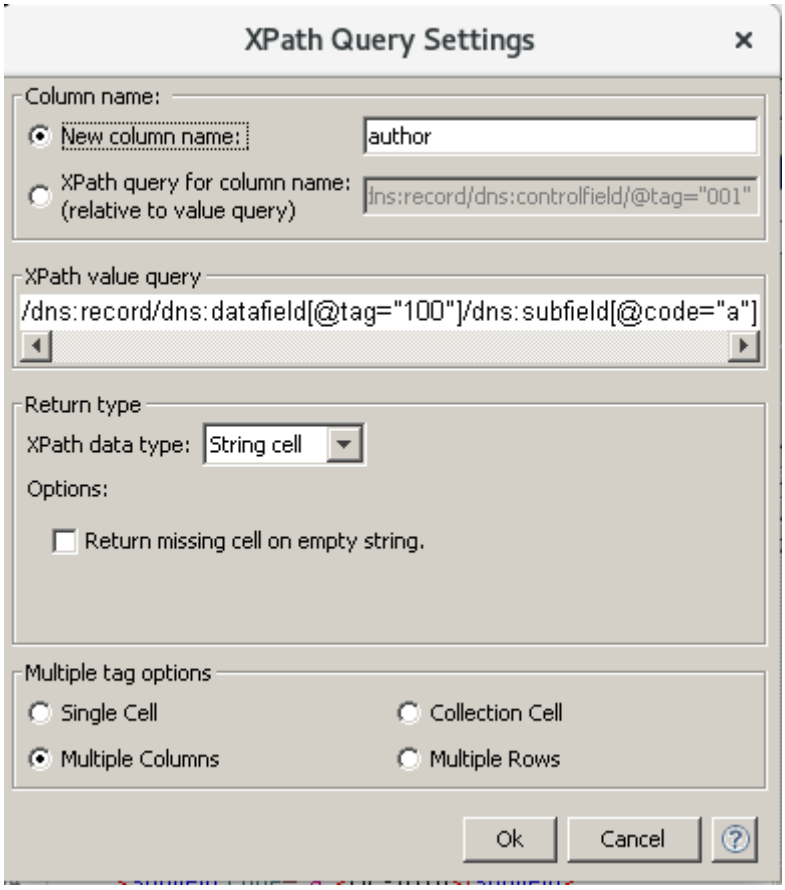


Konfiguration XML Reader und CSV Writer: siehe Anhang B.1

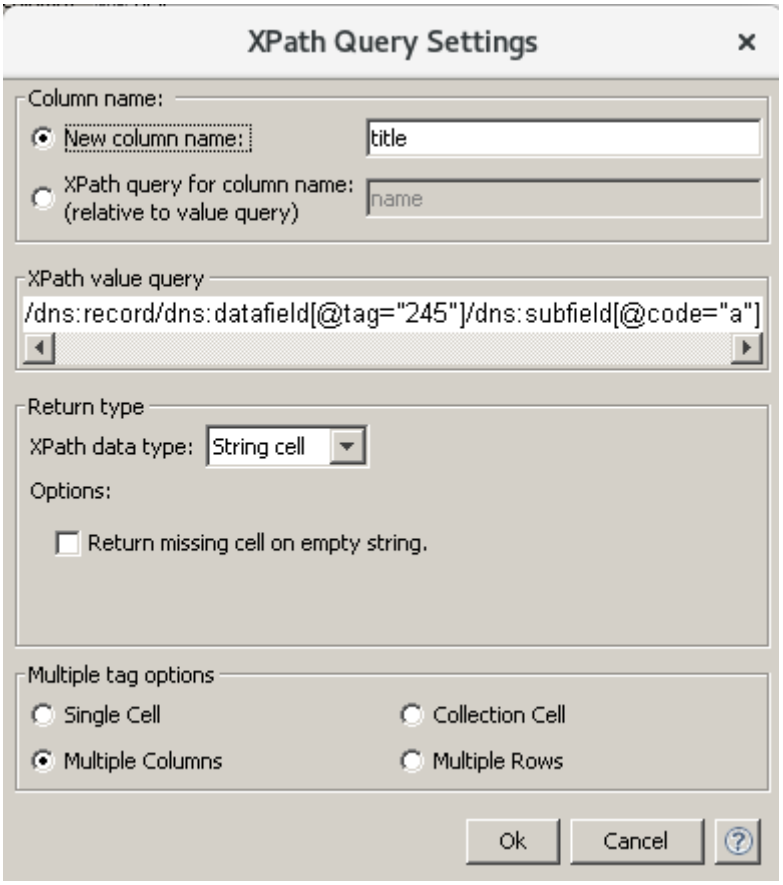
Konfiguration XPath:



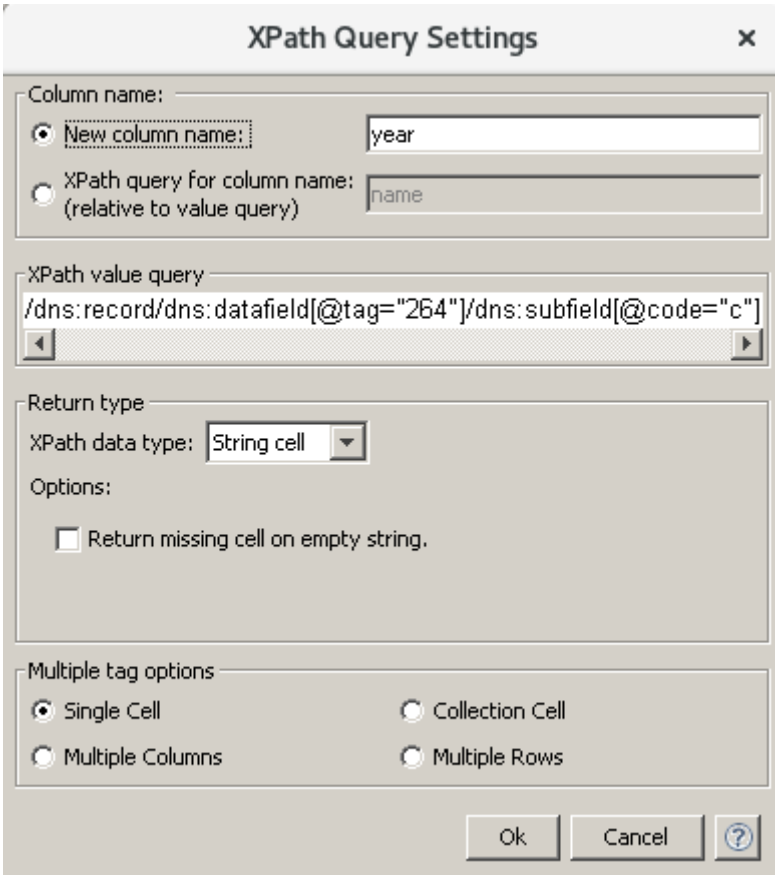
Konfiguration der „author“-Query innerhalb des Xpath-Nodes:



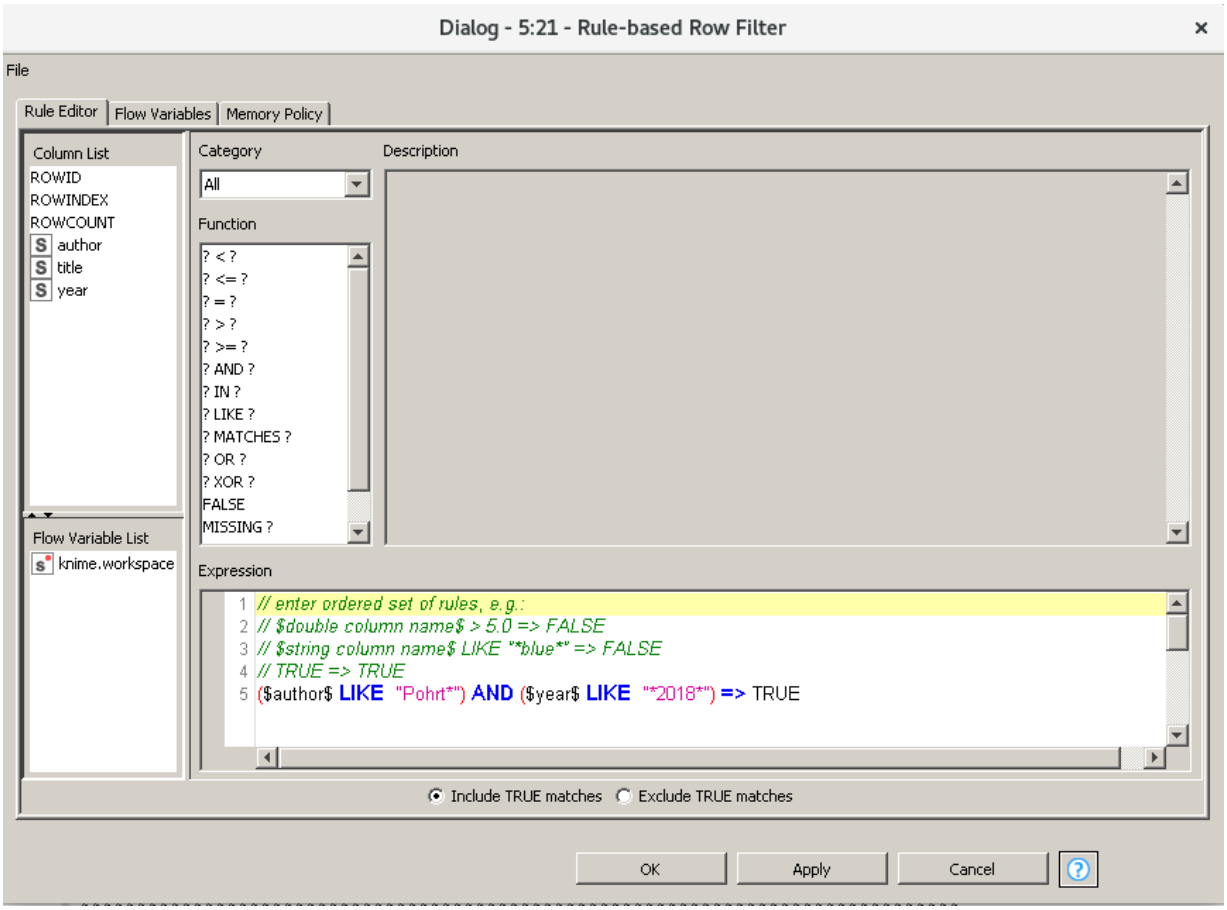
Konfiguration der „title“-Query innerhalb des Xpath-Nodes:



Konfiguration der „year“-Query innerhalb des Xpath-Nodes:

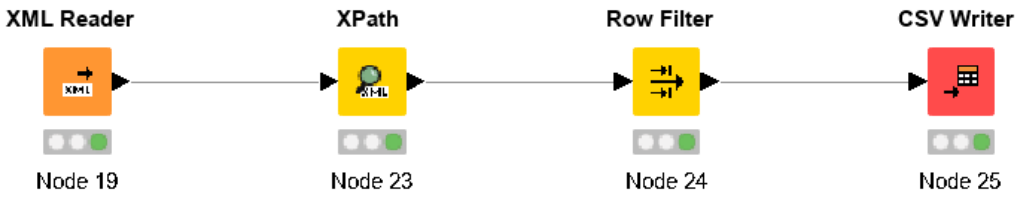


Konfiguration Rule-based Row Filter:



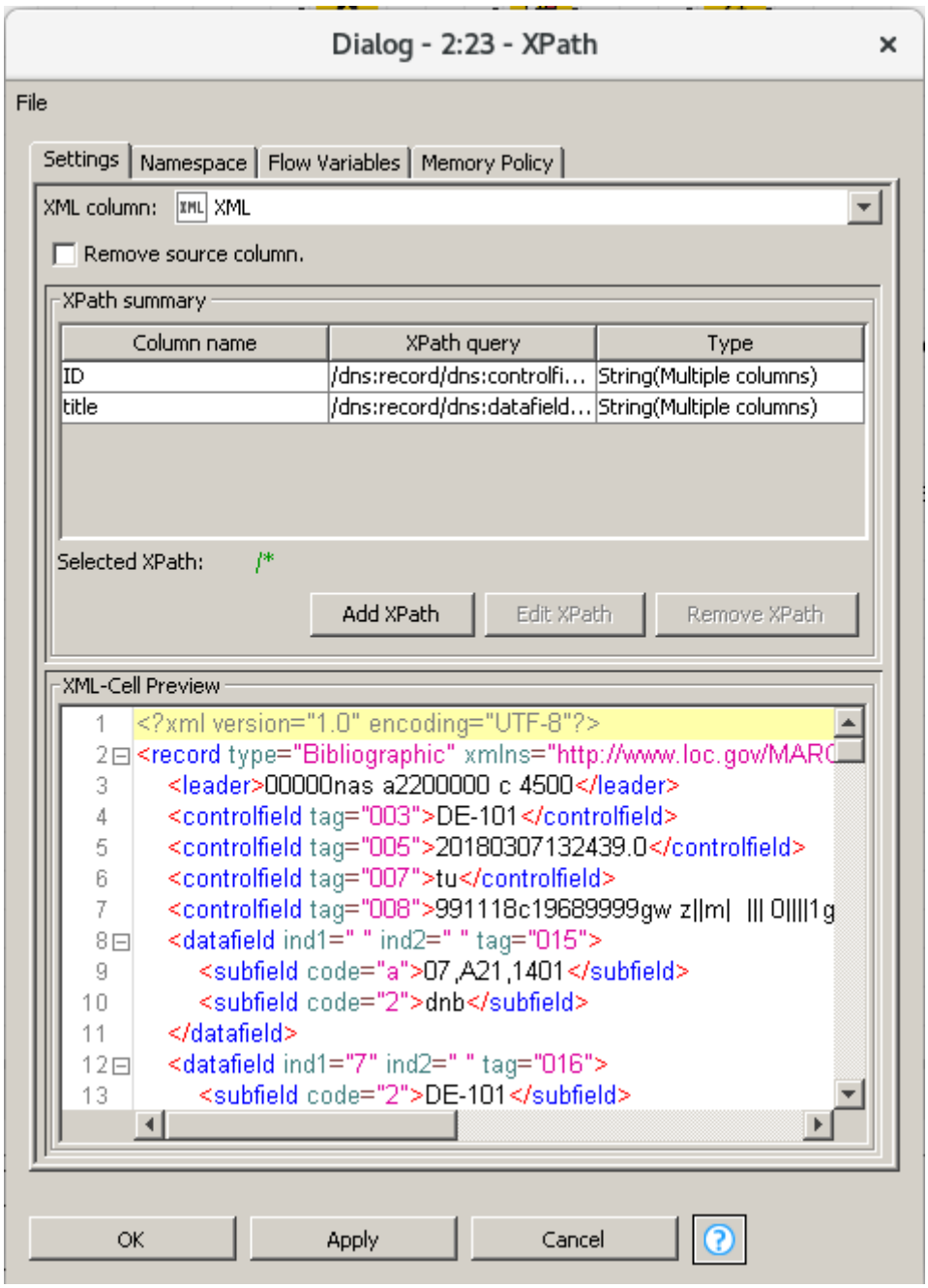
B.3 Analyse: Einträge ohne ID finden

Übersicht:

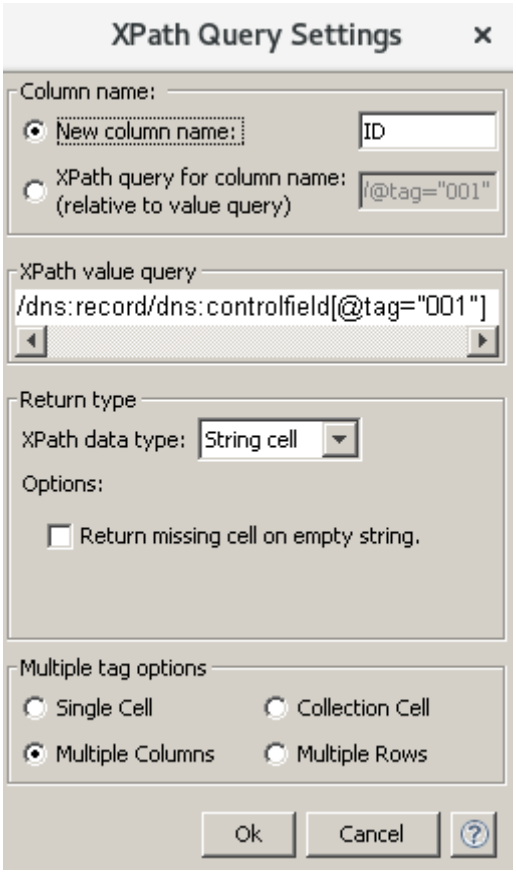


Konfiguration XML Reader und CSV Writer: siehe Anhang B.1

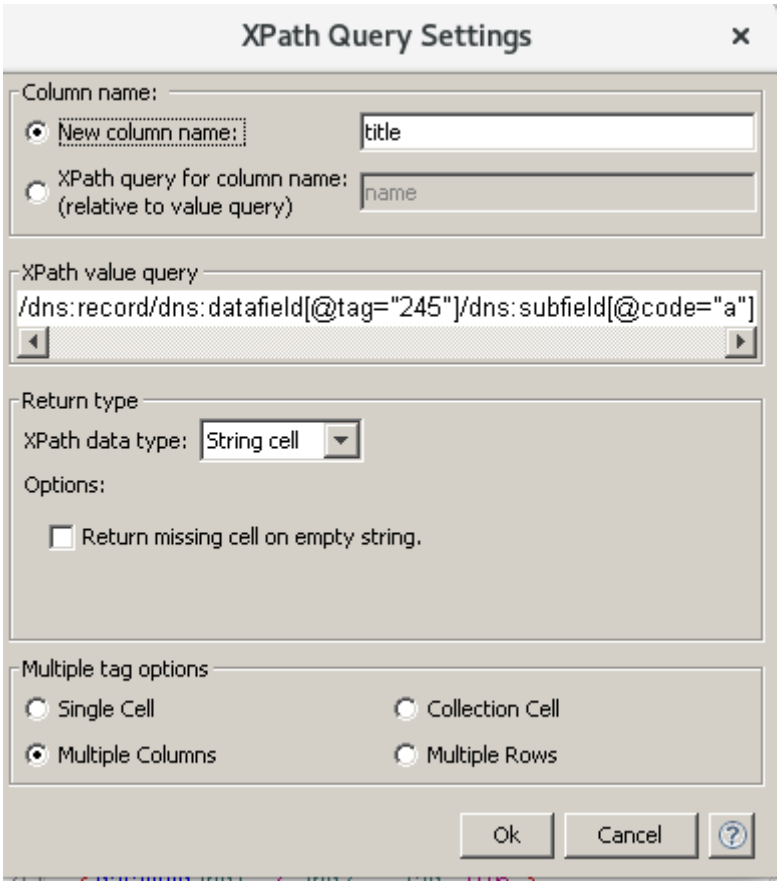
Konfiguration XPath:



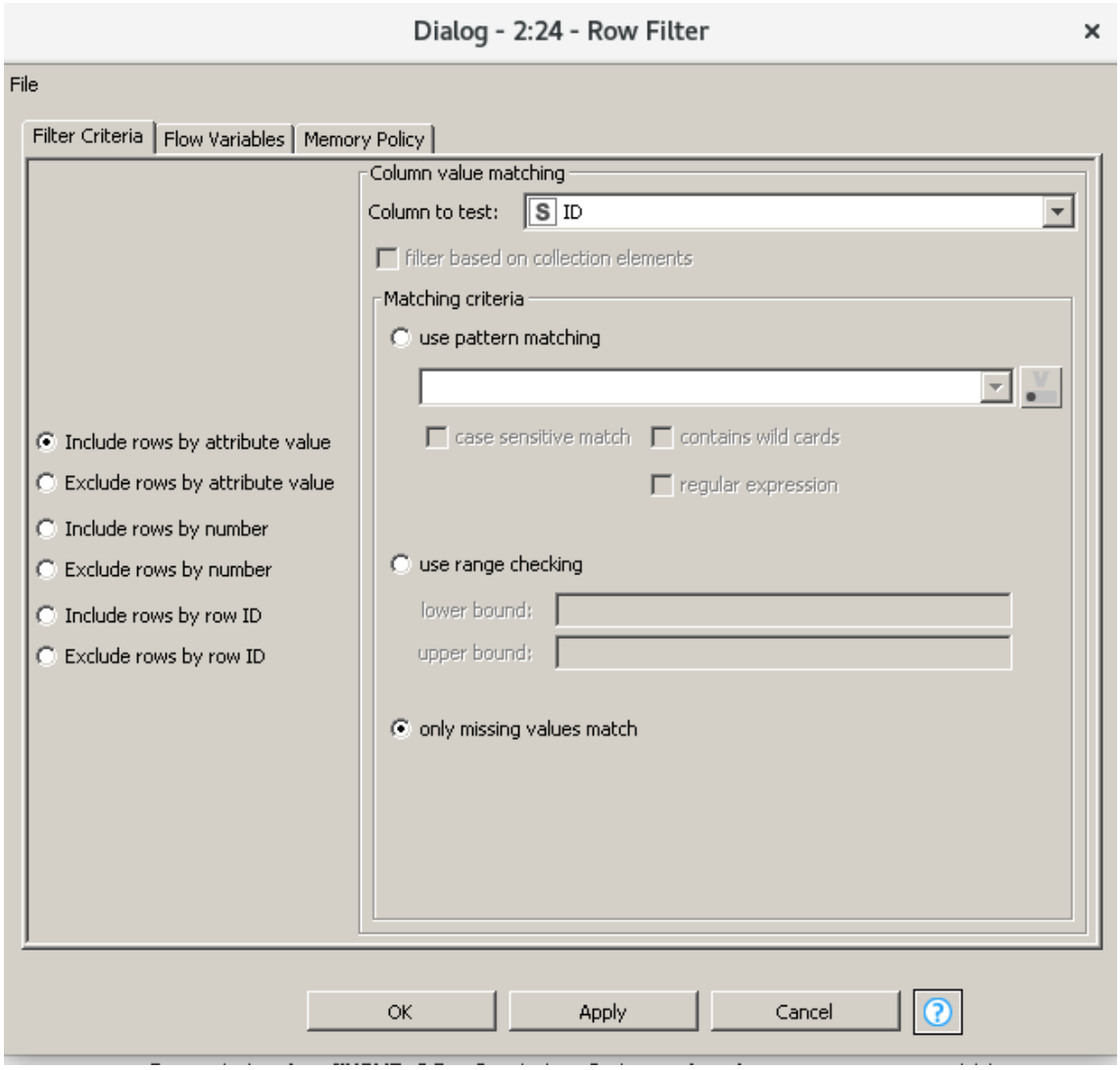
Konfiguration der „ID“-Query innerhalb des Xpath-Nodes:



Konfiguration der „title“-Query innerhalb des Xpath-Nodes:

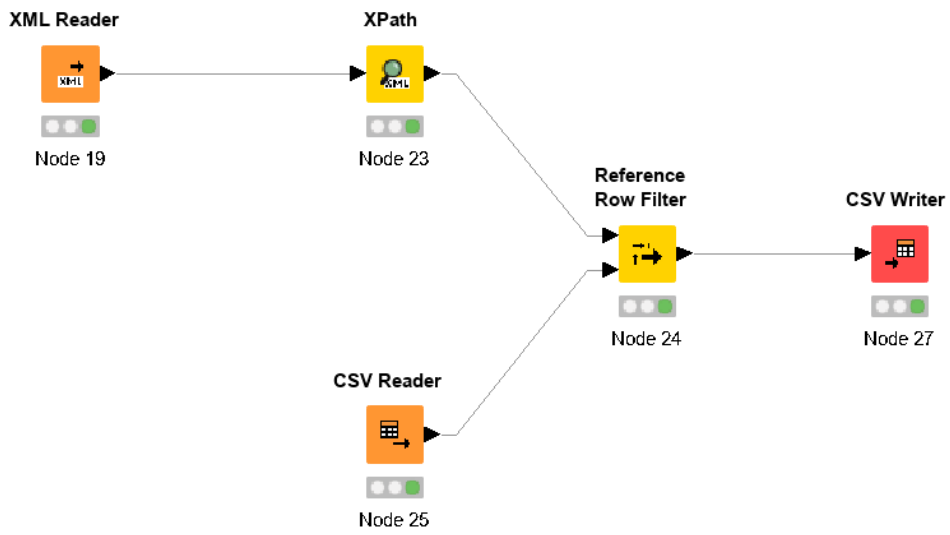


Konfiguration Row Filter:



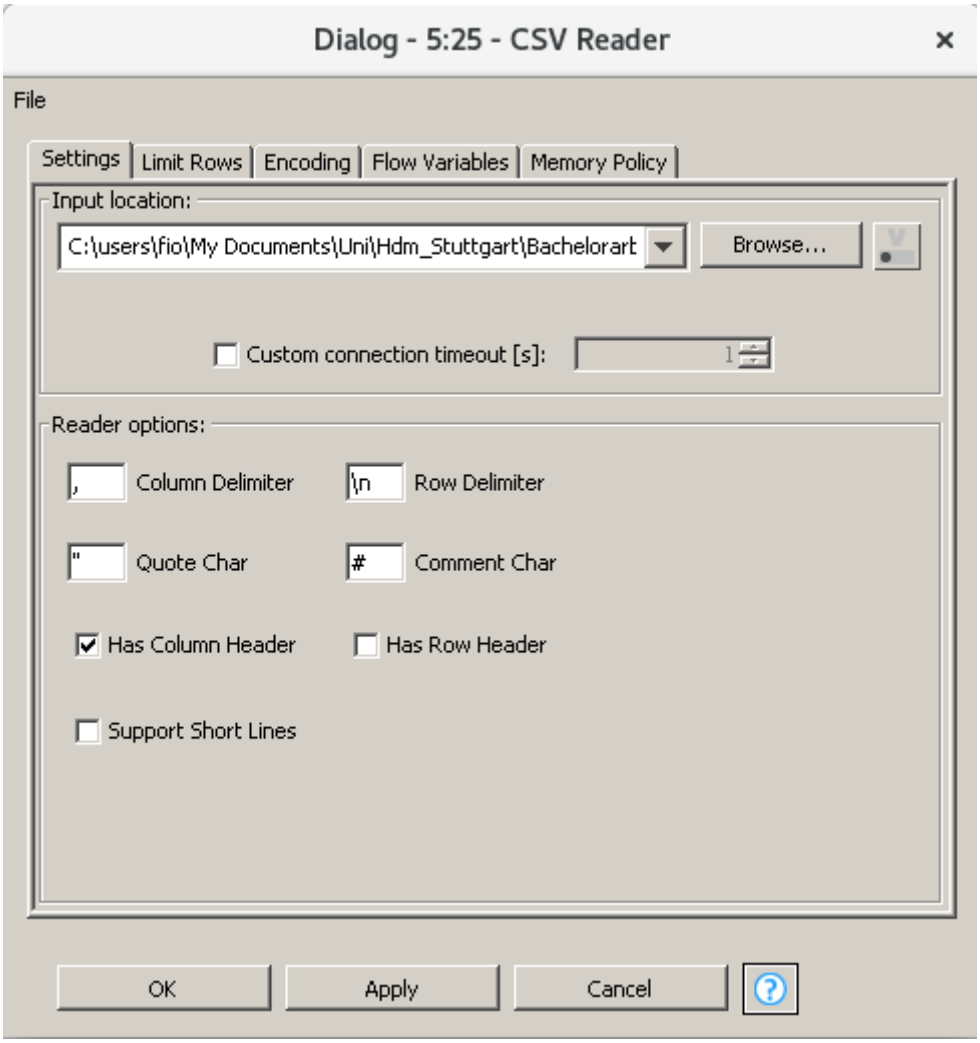
B.4 Analyse: Einträge mit falschen Codes im Datenträgertyp finden

Übersicht:

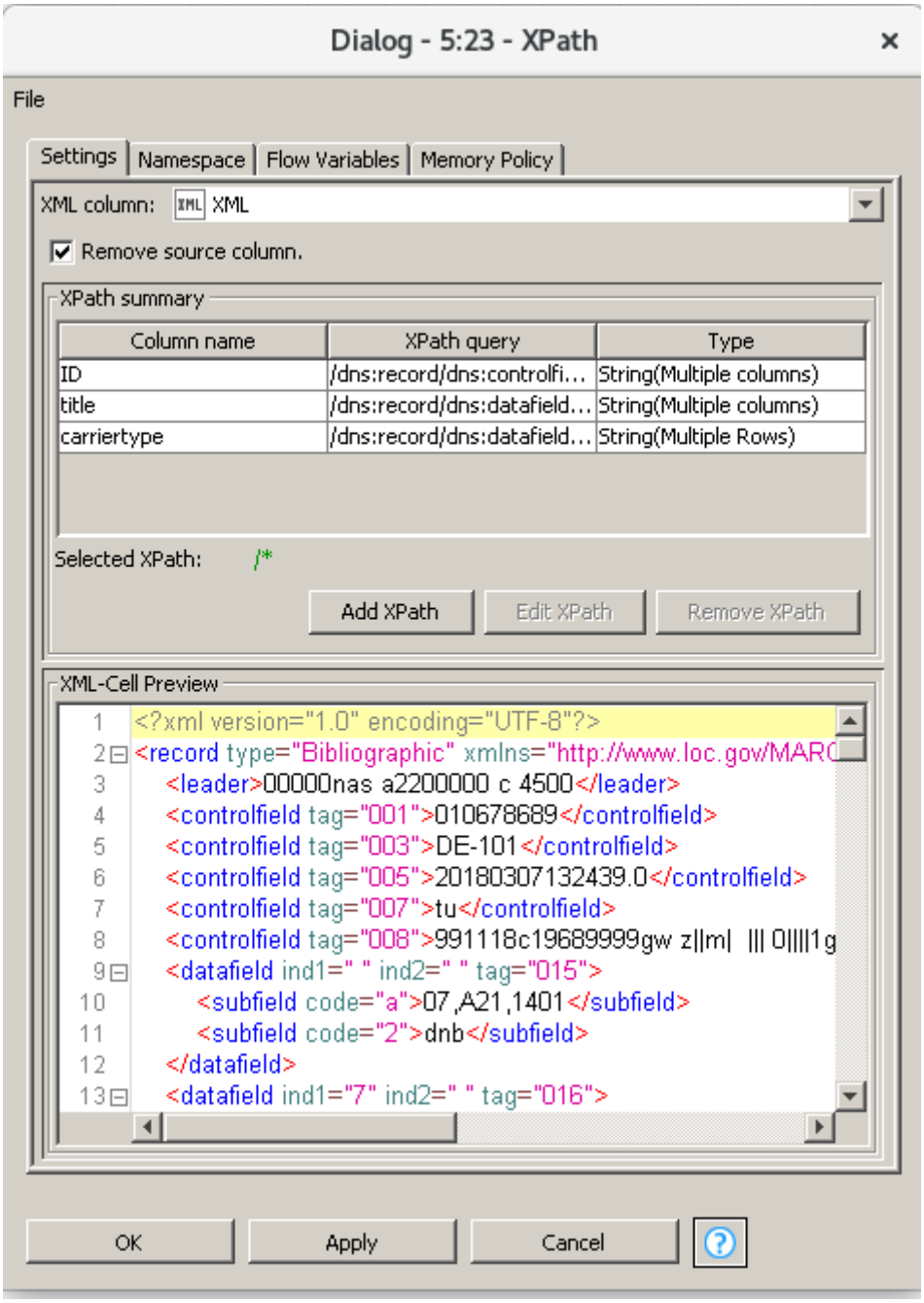


Konfiguration XML Reader und CSV Writer: siehe Anhang B.1

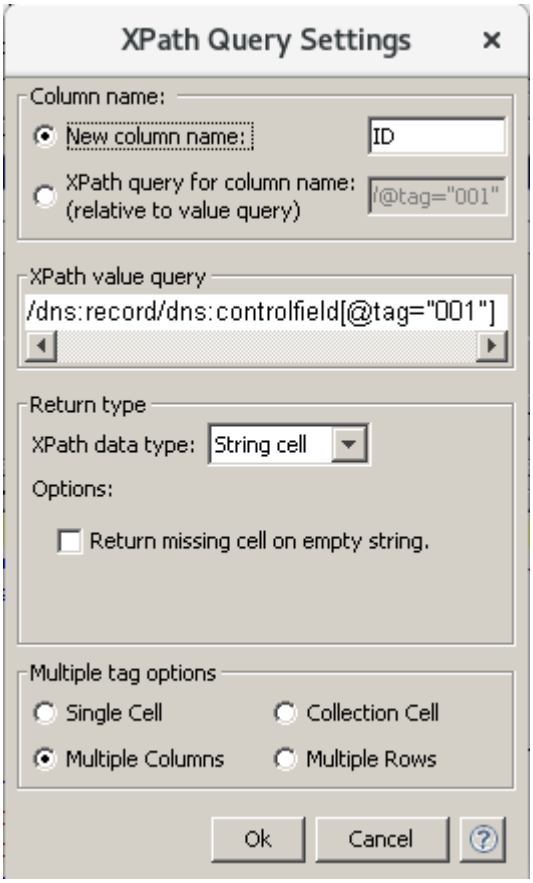
Konfiguration CSV Reader:



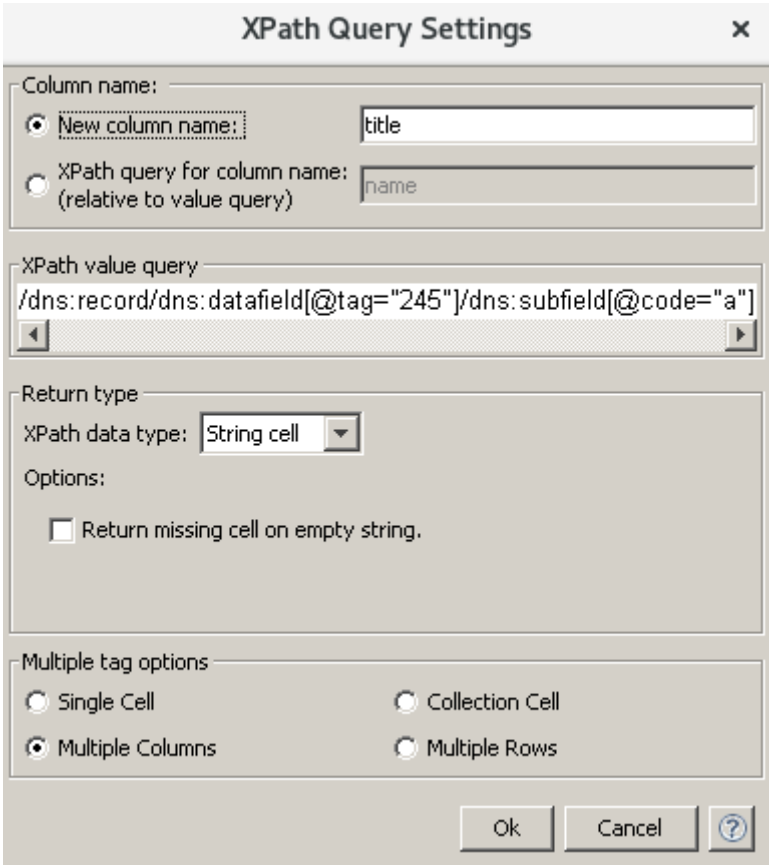
Konfiguration XPath:



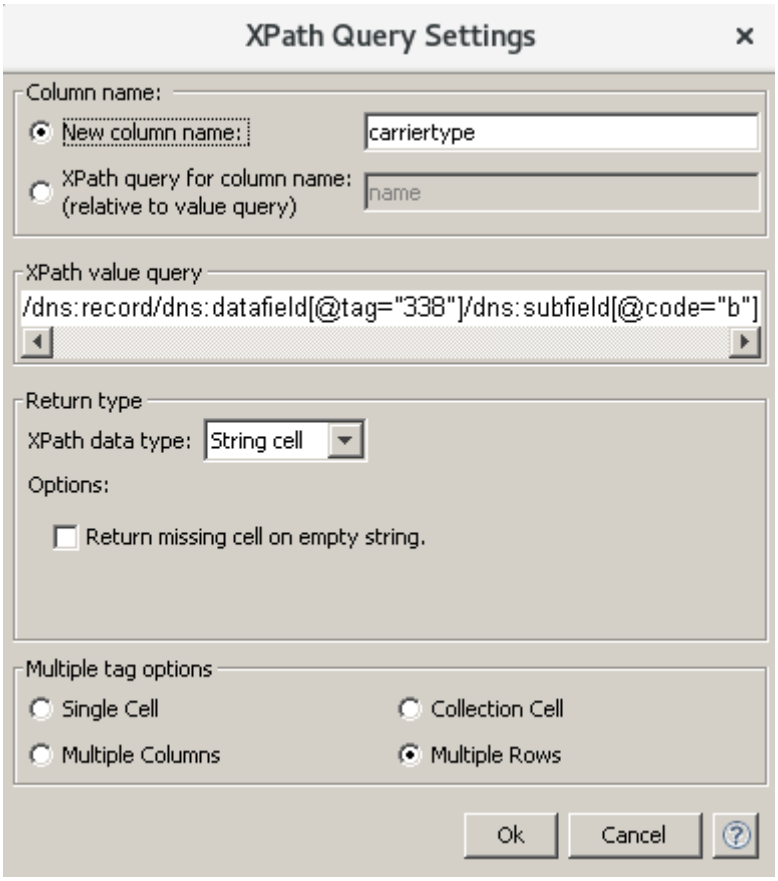
Konfiguration der „ID“-Query innerhalb des Xpath-Nodes:



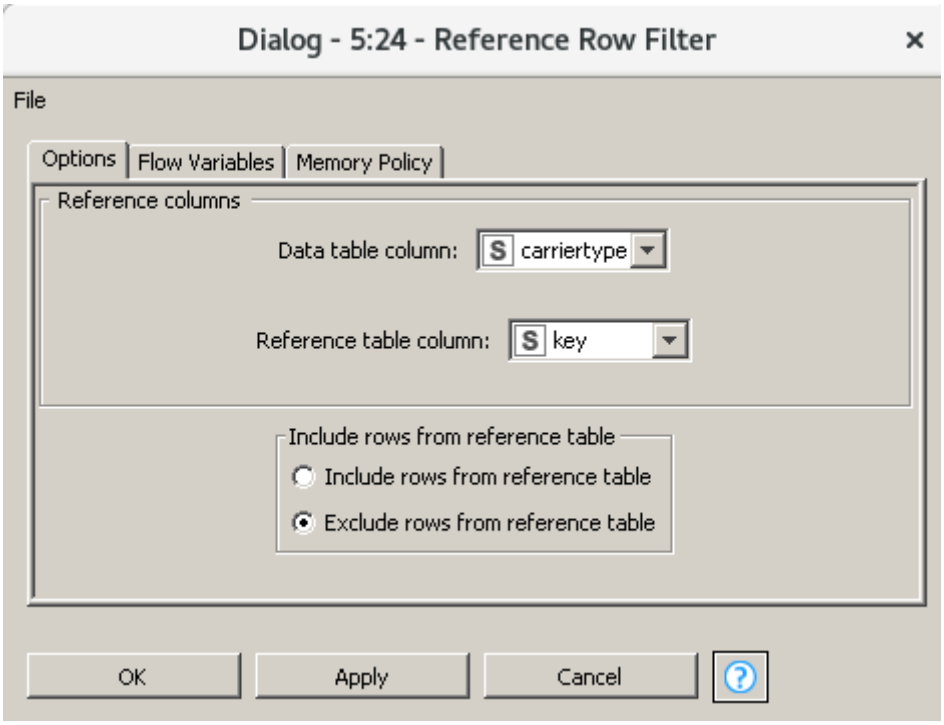
Konfiguration der „title“-Query innerhalb des Xpath-Nodes:



Konfiguration der „carriertype“-Query innerhalb des Xpath-Nodes:

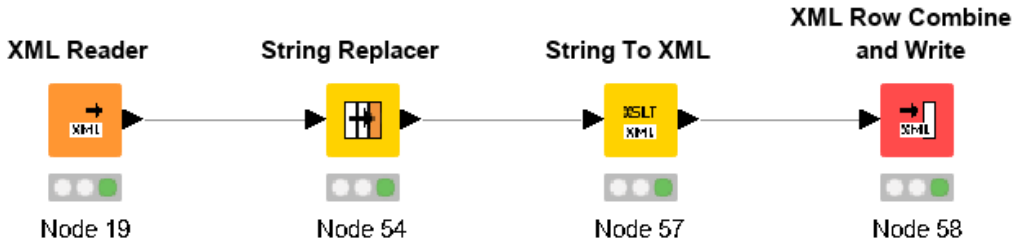


Konfiguration Reference Row Filter:



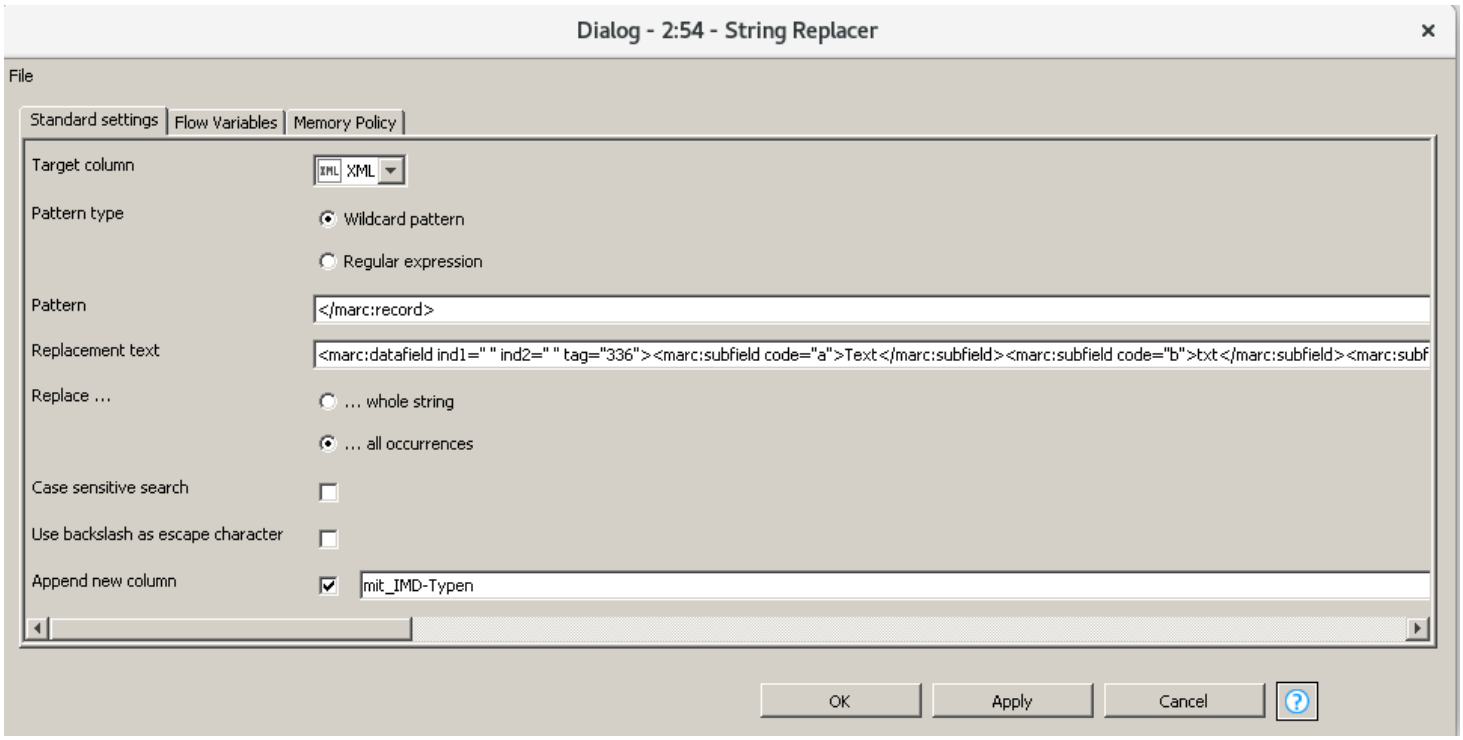
B.5 Ergänzen von IMD-Typen

Übersicht:

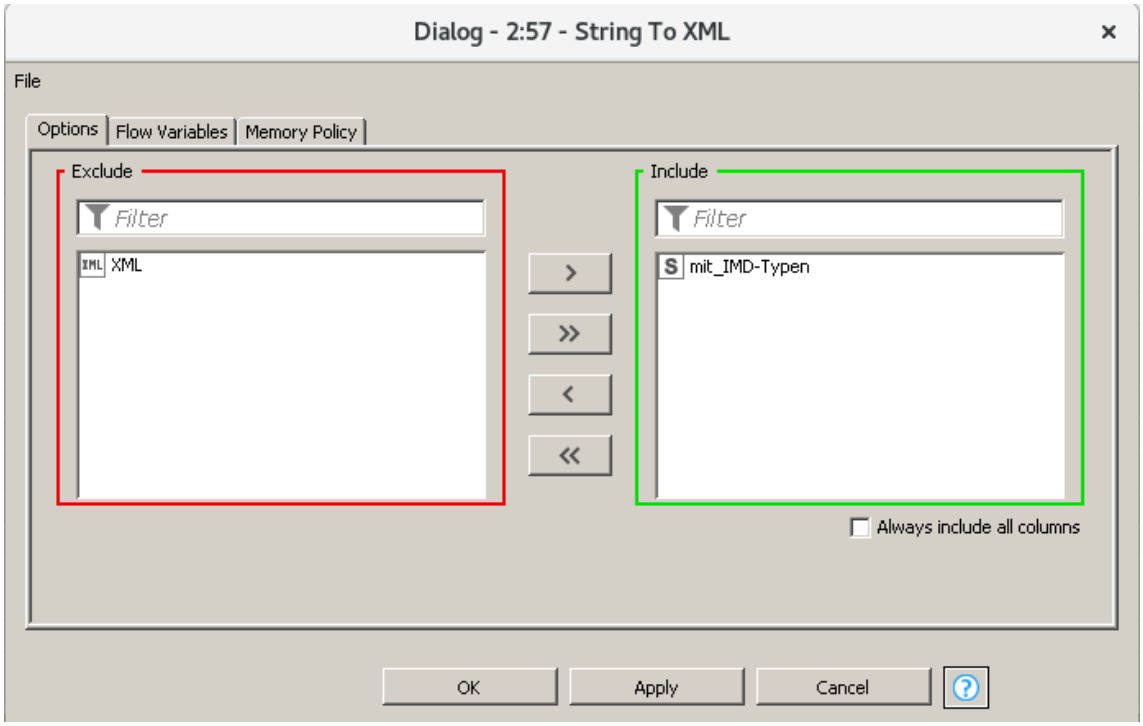


Konfiguration XML Reader: siehe Anhang B.1

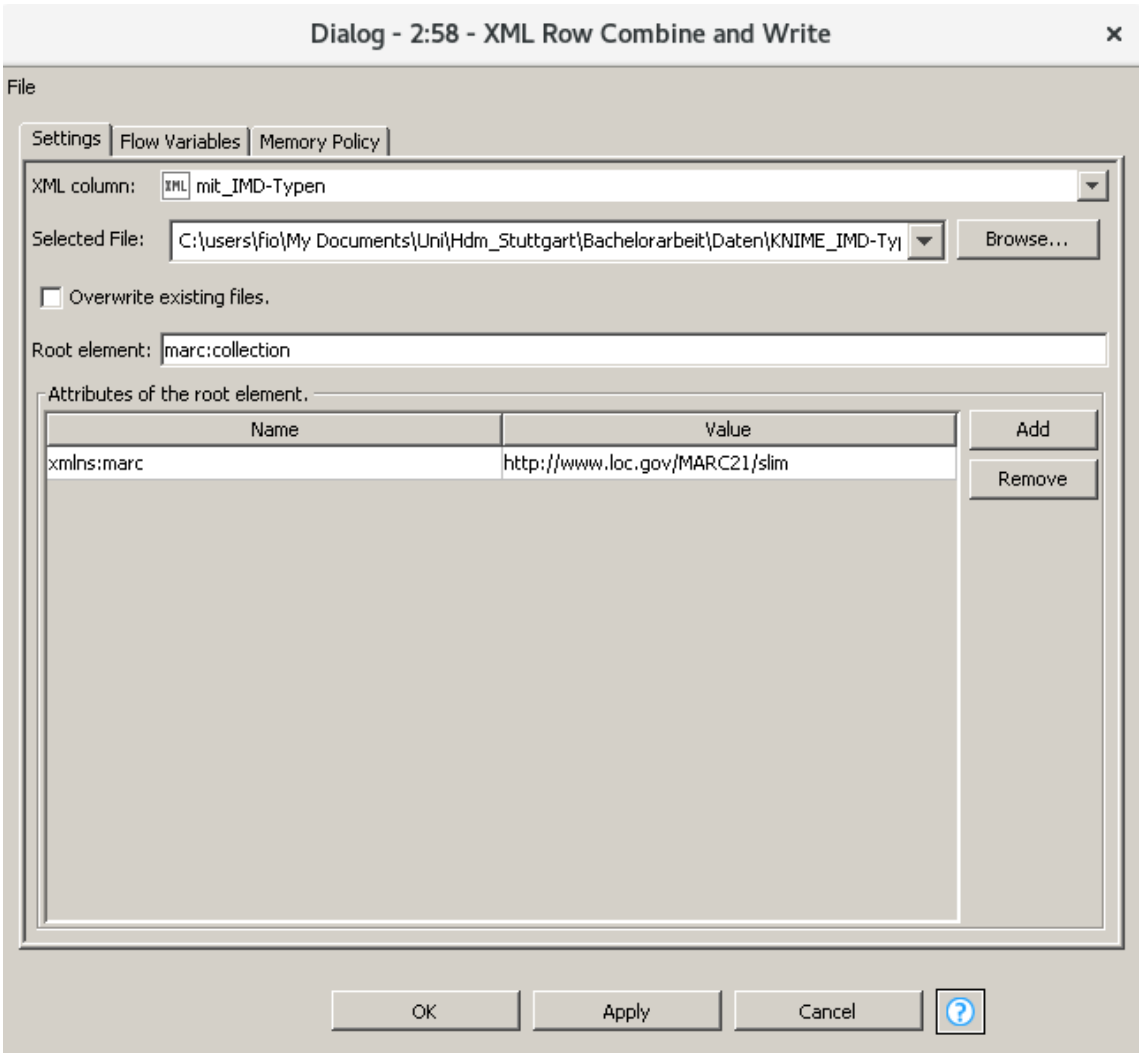
Konfiguration String Replacer:



Konfiguration String To XML:

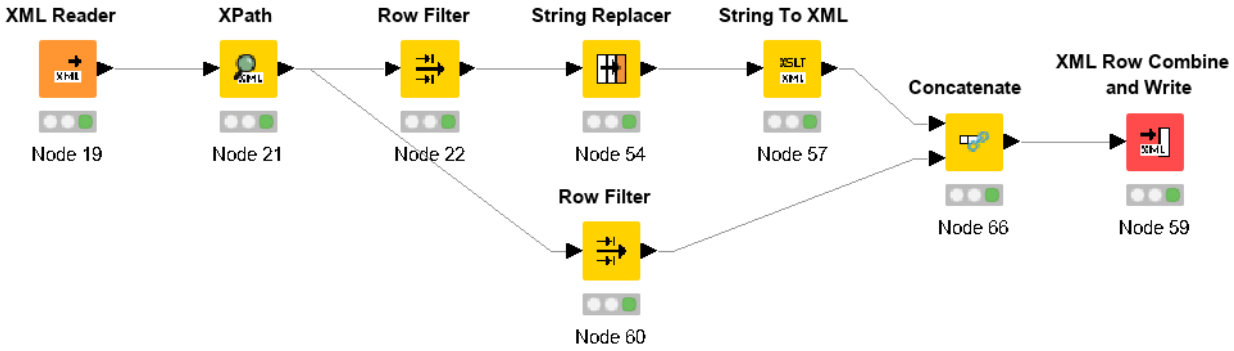


Konfiguration XML Row Combine and Write:



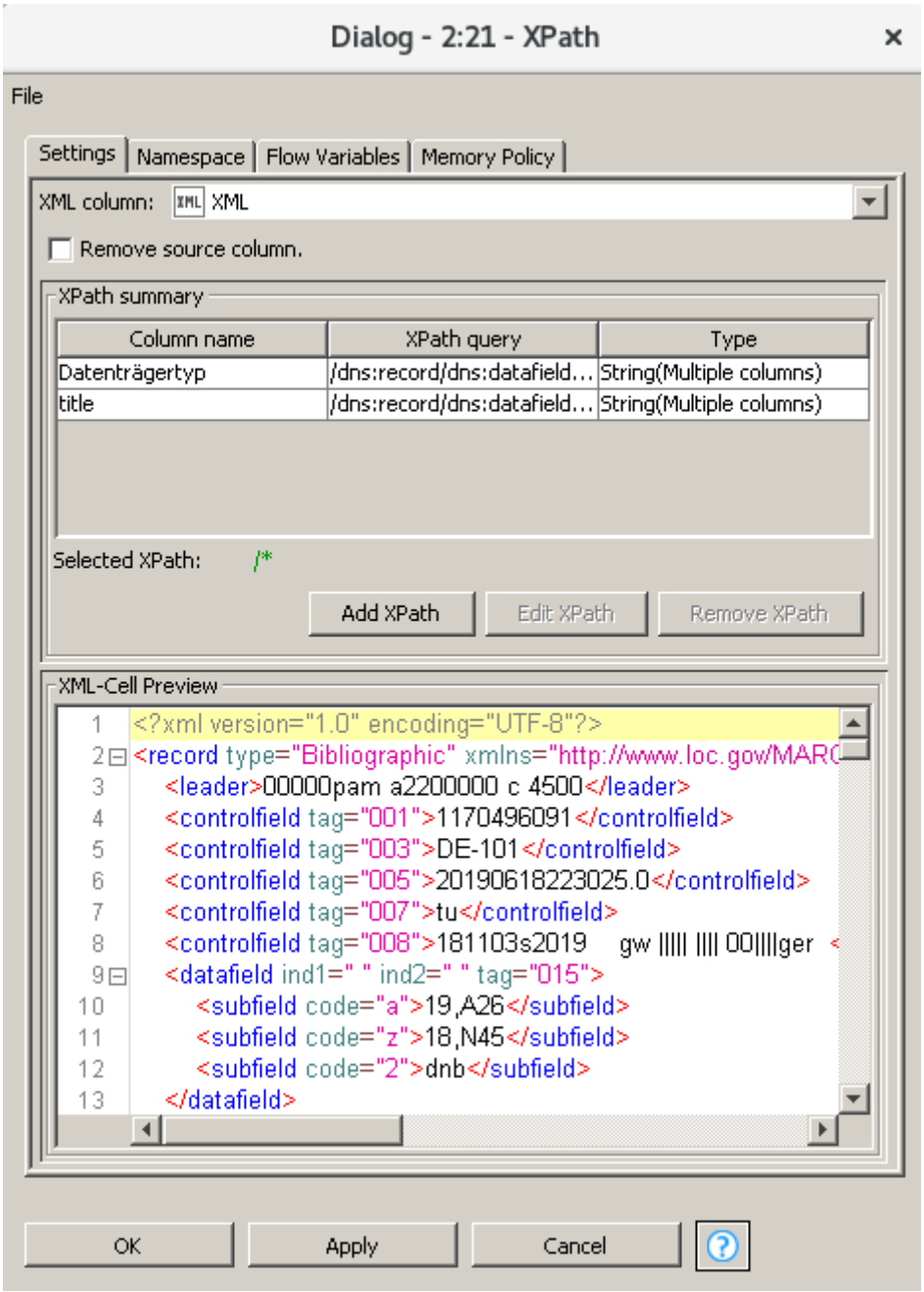
B.6 Ergänzen des Campusnetz-Hinweises bei E-Books

Übersicht:

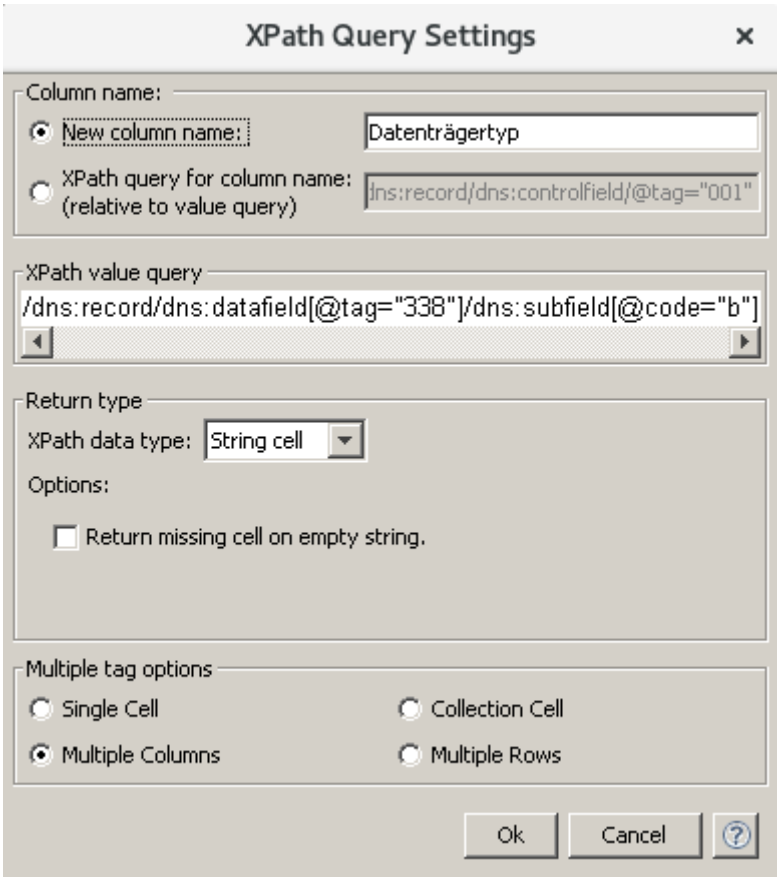


Konfiguration XML Reader: siehe Anhang B.1

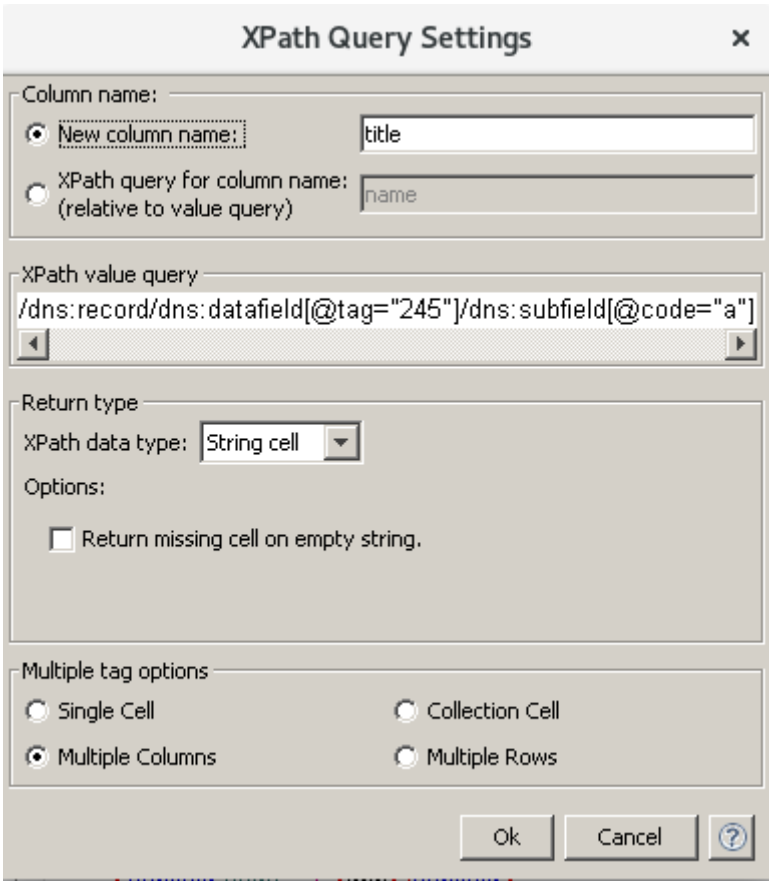
Konfiguration XPath:



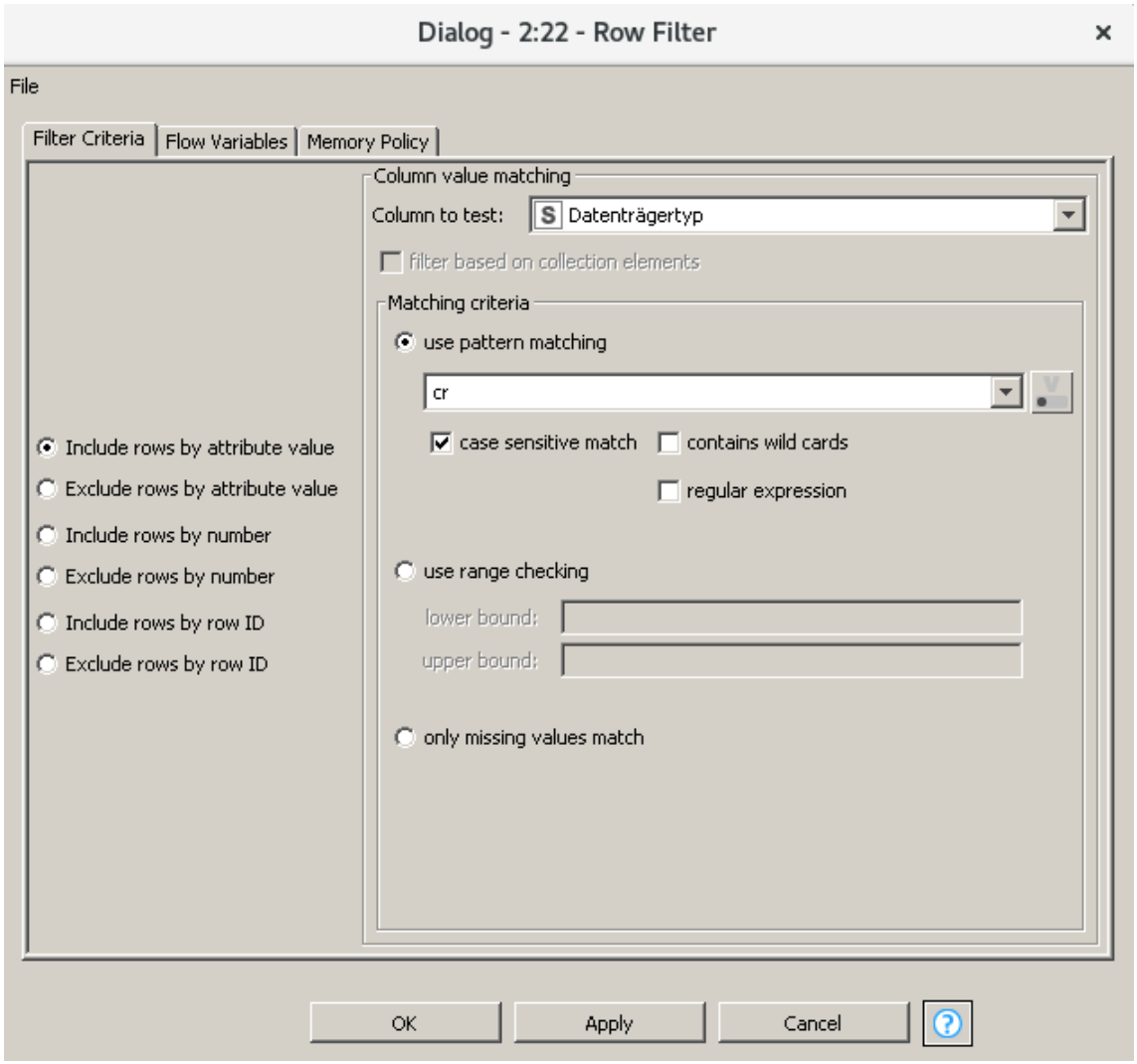
Konfiguration der „Datenträgertyp“-Query innerhalb des Xpath-Nodes:



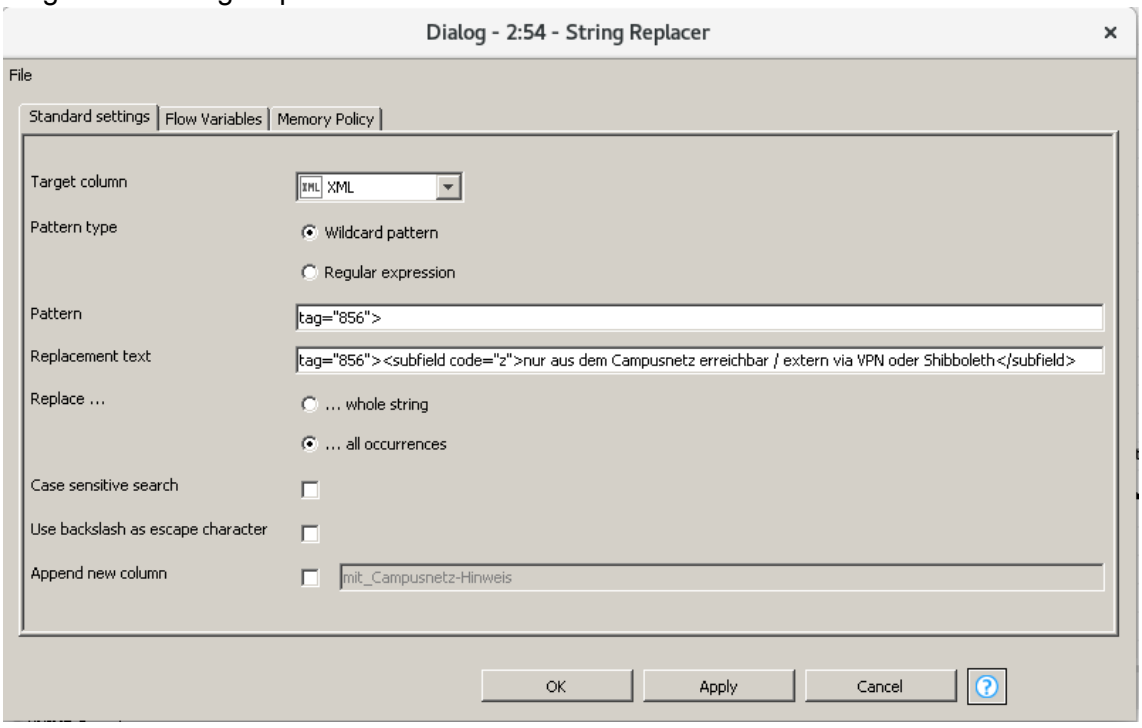
Konfiguration der „title“-Query innerhalb des Xpath-Nodes:



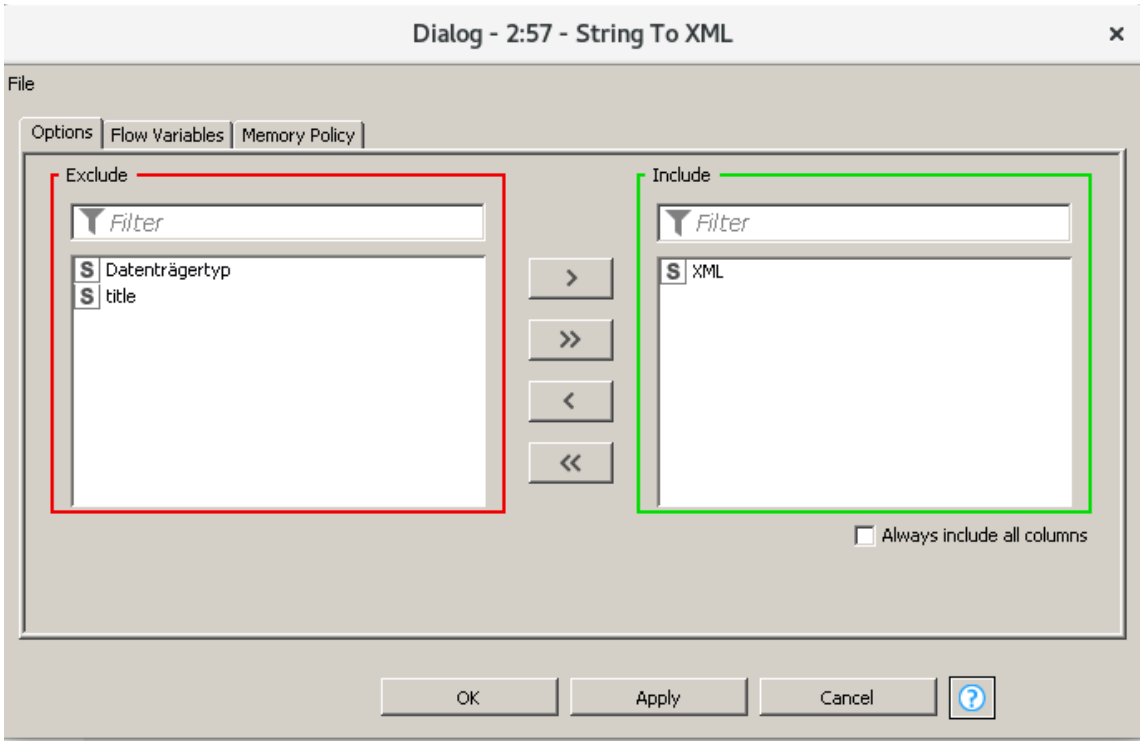
Konfiguration Row Filter (Node 22):



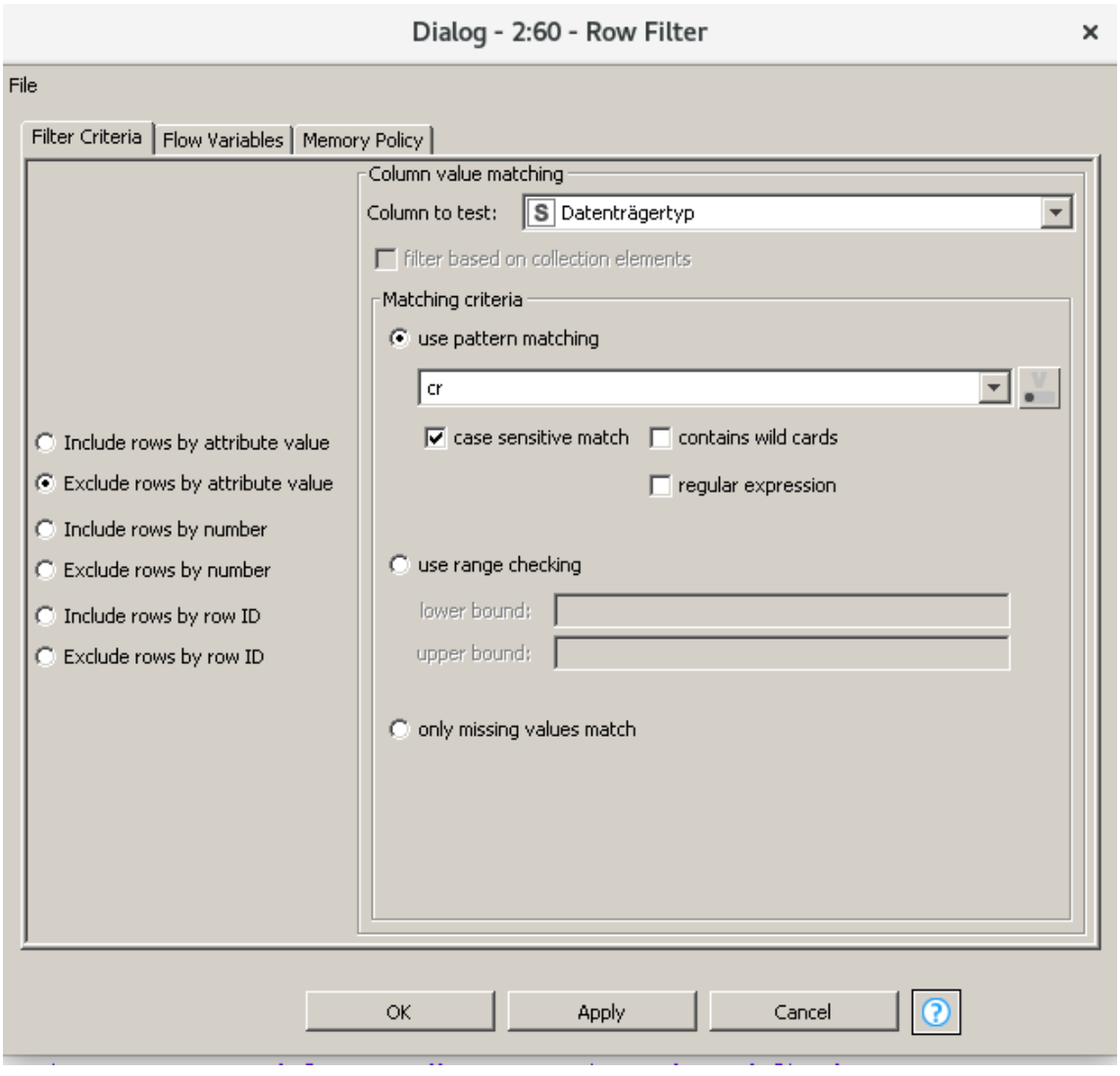
Konfiguration String Replacer:



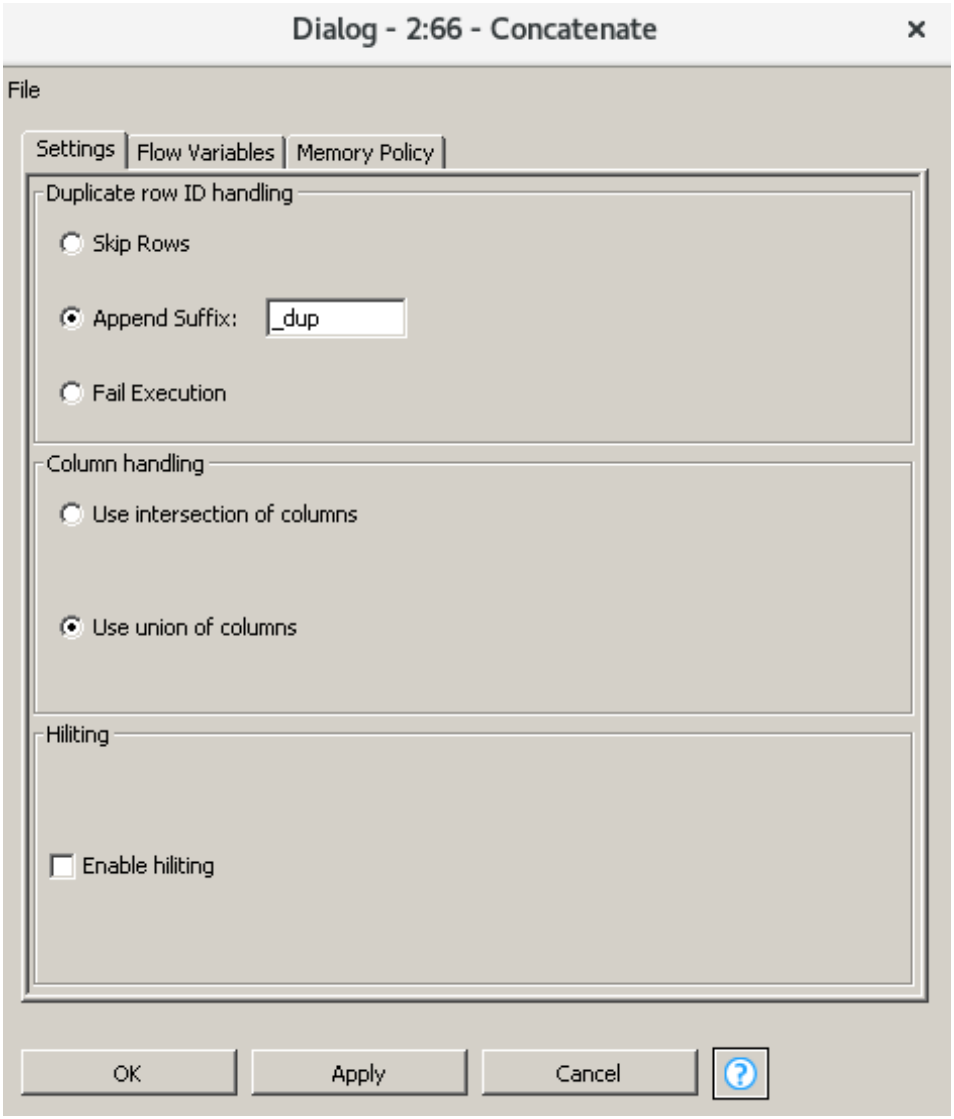
Konfiguration String To XML:



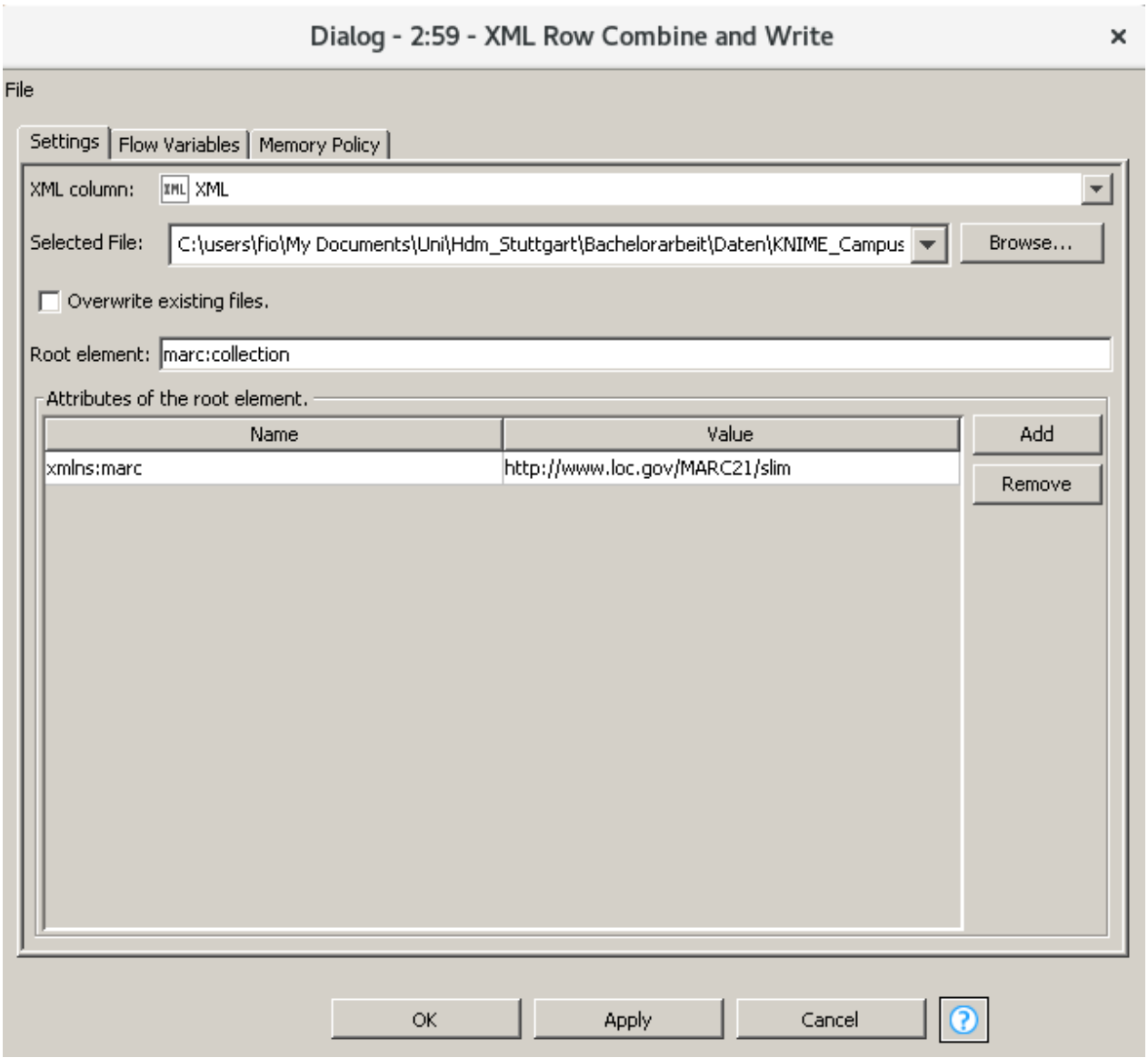
Konfiguration Row Filter (Node 60):



Konfiguration Concatenate:



Konfiguration XML Row Combine and Write:



Quellenverzeichnis

5.1.2 MARC, MAB, PICA paths. (o. J.). Abgerufen 8. November 2019, von http://librecat.org/Catmandu/#marc-mab-pica-paths

About LibreCat — Open Tools for Library and Research Services. (o. J.). Abgerufen 24. November 2019, von https://librecat.org/about.html

APIs and protocols supported by Koha - Koha Wiki. (2019, Oktober 18). Abgerufen 24. November 2019, von https://wiki.koha-community.org/wiki/APIs_and_protocols_supported_by_Koha

Bash-Skripting-Guide für Anfänger › Shell › Wiki › ubuntuusers.de. (2019, Oktober 30). Abgerufen 24. November 2019, von https://wiki.ubuntuusers.de/Shell/Bash-Skripting-Guide_f%C3%BCr_Anf%C3%A4nger/

Bissegger, J., & Wittwer, B. (2016, Juni). *Metadatenmanagement: die ETH-Bibliothek beschreitet neue Wege*. Education. Abgerufen von https://www.slideshare.net/ETH-Bibliothek/metadatenmanagement-die-ethbibliothek-beschreitet-neue-wege

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., & Yergeau, F. (2013a, Februar 7). Extensible Markup Language (XML) 1.0 (Fifth Edition). Abgerufen 23. November 2019, von https://www.w3.org/TR/REC-xml/#sec-logical-struct

Bray, T., Paoli, J., Sperberg-McQueen, C. M., Maler, E., & Yergeau, F. (2013b, Februar 7). Extensible Markup Language (XML) 1.0 (Fifth Edition). Abgerufen 23. November 2019, von https://www.w3.org/TR/REC-xml/#sec-documents

Build a workflow | KNIME. (o. J.). Abgerufen 24. November 2019, von https://www.knime.com/getting-started

Catmandu. (o. J.-a). Abgerufen 15. November 2019, von https://librecat.org/Catmandu/#introduction

Catmandu. (o. J.-b). Abgerufen 13. November 2019, von http://librecat.org/Catmandu/#installation

Catmandu. (o. J.-c). Abgerufen 24. November 2019, von https://librecat.org/Catmandu/#concepts

Catmandu. (o. J.-d). Abgerufen 24. November 2019, von https://librecat.org/Catmandu/#fixes

Catmandu. (o. J.-e). Abgerufen 13. November 2019, von http://librecat.org/Catmandu/#development-setup

Catmandu - a data toolkit. (o. J.). Abgerufen 7. November 2019, von http://librecat.org/Catmandu/

Catmandu::MARC::Tutorial - A documentation-only module for new users of Catmandu::MARC - metacpan.org. (o. J.). Abgerufen 24. November 2019, von https://metacpan.org/pod/distribution/Catmandu-MARC/lib/Catmandu/MARC/Tutorial.pod

Cavegn-Pfister, E., Wirth, A., Wittwer, B., & Wolff, M. (2017). Metadatenmanagement – Wie die ETH-Bibliothek ein neu entstandenes Arbeitsfeld bedient. *Arbido*, (3). Abgerufen von https://www.research-collection.ethz.ch/handle/20.500.11850/201746

Chen, Y.-N., Wen, C.-Y., Chen, H.-P., Lin, Y.-H., & Sum, H.-C. (2011). Metrics for Metadata Quality Assurance and Their Implications for Digital Libraries. In C. Xing, F. Crestani, & A. Rauber (Hrsg.), *Digital Libraries: For Cultural Heritage, Knowledge Dissemination, and Future Creation* (S. 138–147). Springer Berlin Heidelberg.

Cole, T. W., & Foulonneau, M. (2007). *Using the Open Archives Initiative protocol for metadata harvesting*. Westport, Conn. [u.a.]: Libraries Unlimited.

Cole, T. W., Han, M.-J. K., & Schwartz, C. (2018). *Coding with XML for efficiencies in cataloging and metadata: practical applications of XSD, XSLT, and XQuery*. Chicago: ALA Editions.

Create a New KNIME Extension: Quickstart Guide. (o. J.). Abgerufen 12. November 2019, von https://docs.knime.com/2019-06/analytics_platform_new_node_quickstart_guide/index.html

Day 15 : MARC to Dublin Core. (2014, Dezember 19). Abgerufen 25. November 2019, von Catmandu website: https://librecatproject.wordpress.com/2014/12/19/day-xx-marc-to-dublin-core-12/

DCMI: DCMI Usage Board. (2019, November 19). Abgerufen 21. November 2019, von https://www.dublincore.org/groups/usage-board/

DCMI: Recommendation. (2019, November 19). Abgerufen 21. November 2019, von https://www.dublincore.org/specification_status/recommendation/

Developers | KNIME. (o. J.). Abgerufen 12. November 2019, von https://www.knime.com/developers

Distributions. (o. J.). Abgerufen 22. November 2019, von https://librecat.org/distributions.html

Download | Catmandu. (o. J.). Abgerufen 13. November 2019, von https://librecatproject.wordpress.com/get-catmandu/

Download KNIME Analytics Platform | KNIME. (o. J.). Abgerufen 12. November 2019, von https://www.knime.com/downloads/download-knime

FAQ | KNIME. (o. J.). Abgerufen 23. Juli 2019, von https://www.knime.com/faq#q17

FAQ | KNIME. (o. J.-a). Abgerufen 25. November 2019, von https://www.knime.com/faq

FAQ | KNIME. (o. J.-b). Abgerufen 15. November 2019, von https://www.knime.com/faq#q3

Features - Fedora. (o. J.). Abgerufen 24. November 2019, von Duraspace.org website: https://duraspace.org/fedora/about/features/

Furrie, B. (2009). Understanding MARC Bibliographic: Parts 1 to 6. Abgerufen 29. August 2019, von WHAT IS A MARC RECORD, AND WHY IS IT IMPORTANT? website: https://www.loc.gov/marc/umb/um01to06.html

Haake, E., & Opitz, D. (2017). *Because the night... is full of errors - Effizientes Metadatenmanagement mit Nightwatch*. Abgerufen von https://opus4.kobv.de/opus4-bib-info/frontdoor/index/index/docId/3093

Harlow, C. (2015). Data Munging Tools in Preparation for RDF: Catmandu and LODRefine. *The Code4Lib Journal*, (30). Abgerufen von http://journal.code4lib.org/articles/11013

Haynes, D. (2018). *Metadata for information management and retrieval: understanding metadata and its use* (Second edition). London: Facet Publishing.

Hemme, F. (2018, April). *Metadatenmanagement in FOLIO*. https://doi.org/10.5281/zenodo.1228642

Herb, U., Oberländer, A., & Leidinger, T. (2008). Vernetzung von fachlichen und institutionellen Open-Access-Repositorien : Pilotversuch zum Austausch von Metadaten zwischen KOPS, dem institutionellen Repository der Universität Konstanz, und PsyDok, dem fachlichen Repository der Saarländischen Universitäts- und Landesbibliothek im Bereich Psychologie. *Bibliotheksdienst*, *42*(5), 550–555. Abgerufen von https://kops.uni-konstanz.de/handle/123456789/4367

Hipler, G., Prongué, N., & Schneider, R. (2018). Swissbib und linked.swissbib.ch: Leistung und Potenziale einer offenen Plattform für Schweizer Bibliotheksdaten. In Zentralbibliothek Zürich, A. Keller, & S. Uhl (Hrsg.), *Bibliotheken der Schweiz: Innovation durch Kooperation* (S. 160–172). https://doi.org/10.1515/9783110553796-008

Hochstenbach, P. (2018a, November 21). Catmandu::FedoraCommons - Low level Catmandu interface to the Fedora Commons REST API - metacpan.org. Abgerufen 15. November 2019, von https://metacpan.org/pod/Catmandu::FedoraCommons

Hochstenbach, P. (2018b, November 21). Catmandu::Store::FedoraCommons - A Catmandu::Store plugin for the Fedora Commons repository - metacpan.org. Abgerufen 25. November 2019, von https://metacpan.org/pod/Catmandu::Store::FedoraCommons

Hochstenbach, P. (2019a, Mai 20). Catmandu::Importer::OAI - Package that imports OAI-PMH feeds - metacpan.org. Abgerufen 15. November 2019, von https://metacpan.org/pod/Catmandu::Importer::OAI

Hochstenbach, P. (2019b, Mai 20). Catmandu::Store::OAI - A Catmandu store backed by OAI-PMH - metacpan.org. Abgerufen 15. November 2019, von https://metacpan.org/pod/Catmandu::Store::OAI

Hochstenbach, P. (2019c, Juni). *Catmandu Fixes Cheat Sheet*. Abgerufen von https://librecat.org/assets/catmandu_cheat_sheet.pdf

Hochstenbach, P. (2019d, Oktober 17). Catmandu::Fix::marc_add - add new fields to marc - metacpan.org. Abgerufen 27. November 2019, von https://metacpan.org/pod/Catmandu::Fix::marc_add

Hochstenbach, P. (2019e, Oktober 17). Catmandu::Importer::MARC - Package that imports MARC data - metacpan.org. Abgerufen 14. November 2019, von https://metacpan.org/pod/Catmandu::Importer::MARC

Hochstenbach, P. (2019f, Oktober 17). Catmandu-MARC-1.254 - Catmandu modules for working with MARC data - metacpan.org. Abgerufen 14. November 2019, von https://metacpan.org/release/Catmandu-MARC

Index of /testdat. (o. J.). Abgerufen 24. Juli 2019, von https://data.dnb.de/testdat/

Install Extensions and Integrations | KNIME. (o. J.). Abgerufen 12. November 2019, von https://www.knime.com/downloads/update

Instructions for Developers | KNIME. (o. J.). Abgerufen 25. November 2019, von https://www.knime.com/community/developers

Introduction to MongoDB — MongoDB Manual. (o. J.). Abgerufen 24. November 2019, von https://github.com/mongodb/docs/blob/master/source/introduction.txt website: https://docs.mongodb.com/manual/introduction

ISO 15836-1:2017. (2017, Mai). Abgerufen 23. November 2019, von ISO website: http://www.iso.org/cms/render/live/en/sites/isoorg/contents/data/standard/07/13/71339.html

Karadkar, U. P., Altman, A., Breedlove, M., & Matienzo, M. (2016). Introduction to the Digital Public Library of America API. *Proceedings of the 16th ACM/IEEE-CS on Joint Conference on Digital Libraries - JCDL '16*, 285–286. https://doi.org/10.1145/2910896.2925428

Katalogeintrag im SWB mit der PPN 1668655357. (o. J.). Abgerufen von http://swb.bsz-bw.de/DB=2.1/PPNSET?PPN=1668655357&INDEXSET=21

Klute, U. (2018). *ETL-Prozesse für bibliothekarische Metadaten: Die Migration lokaler Katalogisate im GBV*. Abgerufen von https://opus4.kobv.de/opus4-th-wildau/frontdoor/index/index/docId/1227

KNIME Analytics Platform | KNIME. (o. J.). Abgerufen 22. November 2019, von https://www.knime.com/knime-analytics-platform

KNIME Cheat Sheets | KNIME. (o. J.). Abgerufen 25. November 2019, von https://www.knime.com/learning/cheatsheets

KNIME Community Forum. (o. J.). Abgerufen 25. November 2019, von KNIME Community Forum website: https://forum.knime.com/

Knime Config Dialogs/Popups Blank. (2017, März 20). Abgerufen 23. Juli 2019, von KNIME Community Forum website: https://forum.knime.com/t/knime-config-dialogs-popups-blank/6318

KNIME Hub. (o. J.). Abgerufen 25. November 2019, von KNIME Hub website: https://hub.knime.com/

KNIME MongoDB Integration. (o. J.). Abgerufen 14. November 2019, von KNIME Hub website: https://hub.knime.com/knime/extensions/org.knime.features.mongodb/latest

KNIME Open Source Story | KNIME. (o. J.). Abgerufen 12. November 2019, von https://www.knime.com/knime-open-source-story

KNIME REST Client Extension. (o. J.). Abgerufen 14. November 2019, von KNIME Hub website: https://hub.knime.com/knime/extensions/org.knime.features.rest/latest

KNIME Semantic Web. (o. J.). Abgerufen 25. November 2019, von KNIME Hub website: https://hub.knime.com/knime/extensions/org.knime.features.semanticweb/latest

KNIME Server | KNIME. (o. J.). Abgerufen 12. November 2019, von https://www.knime.com/knime-server

KNIME Trusted Partners | KNIME. (o. J.). Abgerufen 12. November 2019, von https://www.knime.com/knime-trusted-partners

KNIME XML-Processing. (o. J.). Abgerufen 25. November 2019, von KNIME Hub website: https://hub.knime.com/knime/extensions/org.knime.features.xml/latest

knime/knime-base. (2019, November 25). Abgerufen 25. November 2019, von https://github.com/knime/knime-base

knime/knime-core. (2019, November 12). Abgerufen 12. November 2019, von https://github.com/knime/knime-core/commits/master

knime/knime-core. (o. J.). Abgerufen 12. November 2019, von GitHub website: https://github.com/knime/knime-core/releases

KNIMETV - YouTube. (o. J.). Abgerufen 25. November 2019, von https://www.youtube.com/user/KNIMETV/about

Koch, G., & Koch, W. (2017). Aggregation und Management von Metadaten im Kontext von Europeana. *Mitteilungen der Vereinigung Österreichischer Bibliothekarinnen und Bibliothekare*, *70*(2), 170–178. https://doi.org/10.31263/voebm.v70i2.1776

Library Carpentry: FAIR Data and Software: Accessible. (o. J.). Abgerufen 24. November 2019, von https://librarycarpentry.org/lc-fair-research/03-accessible/index.html

LibreCat & Catmandu - open source applications for libraries -. (o. J.). Abgerufen 22. November 2019, von https://librecat.org/index.html

LibreCat/Catmandu. (2016a, April 8). Abgerufen 13. November 2019, von GitHub website: https://github.com/LibreCat/Catmandu/wiki/Missing-modules

LibreCat/Catmandu. (2016b, Dezember 8). Abgerufen 25. November 2019, von GitHub website: https://github.com/LibreCat/Catmandu/wiki/Cookbook

LibreCat/Catmandu. (2019a, November 6). Abgerufen 13. November 2019, von https://github.com/LibreCat/Catmandu/commits/dev

LibreCat/Catmandu. (2019b, November 13). Abgerufen 25. November 2019, von https://github.com/LibreCat/Catmandu

LibreCat/Catmandu. (o. J.). Abgerufen 13. November 2019, von GitHub website: https://github.com/LibreCat/Catmandu/releases

librecat-dev. (o. J.). librecat-dev Info Page. Abgerufen 17. November 2019, von https://lists.uni-bielefeld.de/mailman2/cgi/unibi/listinfo/librecat-dev

Linked-Data-Service. (o. J.). Abgerufen 21. November 2019, von Deutsche Nationalbibliothek website: https://www.dnb.de/DE/Professionell/Metadatendienste/Datenbezug/LDS/lds_node.html

Liu, J. (2007). *Metadata and its applications in the digital library: approaches and practices*. Westport, Conn. [u.a.]: Libraries Unlimited.

MARC 21 Format for Bibliographic Data: 336: Content Type (Network Development and MARC Standards Office, Library of Congress). (2017, Dezember 18). Abgerufen 27. November 2019, von https://www.loc.gov/marc/bibliographic/concise/bd336.html

MetaCPAN About - metacpan.org. (o. J.). Abgerufen 24. November 2019, von https://metacpan.org/about

Metadata Manipulation | Library Open-Source Software Registry. (o. J.). Abgerufen 27. November 2019, von https://www.foss4lib.org/package-type/metadata-manipulation

Mitchell, E. (2015). *Metadata Standards and Web Services in Libraries, Archives, and Museums: An Active Learning Resource: An Active Learning Resource*. ABC-CLIO.

Mittelbach, J. (2015). Modernes Datenmanagement: Linked Open Data und die offene Bibliothek. *o-bib. Das offene Bibliotheksjournal / Herausgeber VDB*, *2*(2), 61–73. https://doi.org/10.5282/o-bib/2015H2S61-73

Müller, Smolka, & Severin. (2015, Juli 30). *Modul 2.04: IMD-Typen*. Abgerufen von https://wiki.dnb.de/download/attachments/105260204/Modul_2_04_IMD_PICA.pdf?version=1&modificationDate=1441782650000&api=v2

Network Development and MARC Standards Office, Library of Congress (2019). *MARC 21 Format for Bibliographic Data: Table of Contents*. Abgerufen 23. November 2019, von https://www.loc.gov/marc/bibliographic/

OAI. (o. J.). Open Archives Initiative Protocol for Metadata Harvesting. Abgerufen 11. November 2019, von https://www.openarchives.org/pmh/

Osters, M. (2015). *Kuali OLE-Metadatenmanagement: Überblick und Praxis*. Abgerufen von https://hbz.opus.hbz-nrw.de/opus45-hbz/frontdoor/deliver/index/docId/376/file/PDFA_KualiOLE_Workshop_Metadaten_2015_1_15.pdf

Pfeffer, M. (2016). Open Source Software zur Verarbeitung und Analyse von Metadaten. *Vortrag Auf Dem 6. Bibliothekskongress in Leipzig Am 16.03.2016.* Abgerufen von https://opus4.kobv.de/opus4-bib-info/frontdoor/index/index/docId/2449

Pfister, E., Wittwer, B., & Wolff, M. (2017). Metadaten–Manuelle Datenpflege vs. Automatisieren. *BIT online*, *20*(1), 22–25. Abgerufen von http://www.b-i-t-online.de/heft/2017-01-nachrichtenbeitrag-pfister.pdf

Praetere, P. D. (2017, April 28). Catmandu::Store::REST - Store/retrieve items from a JSON REST-API endpoint - metacpan.org [Catmandu::Store::REST]. Abgerufen 15. November 2019, von https://metacpan.org/pod/Catmandu::Store::REST

Previous Versions | KNIME. (o. J.). Abgerufen 12. November 2019, von https://www.knime.com/download-previous-versions

RDA in MARC - October 2012: MARC Standards (Network Development and MARC Standards Office, Library of Congress). (2017, Juli 25). Abgerufen 23. November 2019, von http://www.loc.gov/marc/RDAinMARC.html

REST OAI-PMH. (2019, April 19). Abgerufen 25. November 2019, von Drupal.org website: https://www.drupal.org/project/rest_oai_pmh

RESTful Webservices (1): Was ist das überhaupt? (o. J.). Abgerufen 25. November 2019, von https://www.mittwald.de/blog/webentwicklung-design/webentwicklung/restful-webservices-1-was-ist-das-uberhaupt

solid IT. (2019, November). DB-Engines Ranking. Abgerufen 16. November 2019, von DB-Engines website: https://db-engines.com/de/ranking/document+store

Steenlant, N. (2019a, Juni 18). Catmandu::Store::MongoDB - A searchable store backed by MongoDB - metacpan.org. Abgerufen 15. November 2019, von https://metacpan.org/pod/Catmandu::Store::MongoDB

Steenlant, N. (2019b, November 4). Catmandu::Fix::Condition::exists - only execute fixes if the path exists - metacpan.org. Abgerufen 7. November 2019, von https://metacpan.org/pod/Catmandu::Fix::Condition::exists

Steenlant, N. (2019c, November 13). [librecat-dev] statictic subfield report. Abgerufen 17. November 2019, von https://lists.uni-bielefeld.de/mailman2/unibi/public/librecat-dev/2019-November/000486.html

Steenlant, N. (2019d, November 26). Catmandu::Fix::Bind::list - a binder that computes Fix-es for every element in a list - metacpan.org. Abgerufen 27. November 2019, von https://metacpan.org/pod/Catmandu::Fix::Bind::list

Stephan, R., & Klettke, M. (2017). *Speicherung bibliographischer Metadaten in relationalen und nicht-relationalen Datenbankmodellen*. https://doi.org/10.18420/in2017_116

Term and Code List for RDA Carrier Types: Value Lists for Codes and Controlled Vocabularies (Network Development and MARC Standards Office, Library of Congress). (2019, Juli 11). Abgerufen 27. November 2019, von http://www.loc.gov/standards/valuelist/rdacarrier.html

The librecat-dev Archives. (o. J.). Abgerufen 25. November 2019, von https://lists.uni-bielefeld.de/mailman2/unibi/public/librecat-dev/

Tutorial. (2019, Mai 22). Abgerufen 25. November 2019, von Catmandu website: https://librecatproject.wordpress.com/tutorial/

Use Cases. (o. J.). Abgerufen 25. Juli 2019, von https://librecat.org/use-cases.html

Voß, J. (2017a, August 29). Catmandu::Exporter::RDF - serialize RDF data - metacpan.org. Abgerufen 15. November 2019, von https://metacpan.org/pod/Catmandu::Exporter::RDF

Voß, J. (2017b, August 29). Catmandu::Fix::aref_query - copy values of RDF in aREF to a new field - metacpan.org. Abgerufen 15. November 2019, von https://metacpan.org/pod/Catmandu::Fix::aref_query

Voß, J. (2017c, August 29). Catmandu::Fix::rdf_ldf_statements - lookup an object into a LDF endpoint - metacpan.org. Abgerufen 15. November 2019, von https://metacpan.org/pod/Catmandu::Fix::rdf_ldf_statements

Voß, J. (2017d, August 29). Catmandu::RDF - Modules for handling RDF data within the Catmandu framework - metacpan.org. Abgerufen 14. November 2019, von https://metacpan.org/pod/Catmandu::RDF

Voß, J. (2017e, August 29). Catmandu-RDF-0.32 - Modules for handling RDF data within the Catmandu framework - metacpan.org. Abgerufen 14. November 2019, von https://metacpan.org/release/Catmandu-RDF

wc man page. (2010, September). Abgerufen 26. November 2019, von http://linuxcommand.org/lc3_man_pages/wc1.html

Westbrooks, E. L. (2005). Remarks on metadata management. *OCLC Systems & Services: International Digital Library Perspectives*. https://doi.org/10.1108/10650750510578073

What Is MongoDB? (o. J.). Abgerufen 24. November 2019, von MongoDB website: https://www.mongodb.com/what-is-mongodb

What is REST? (o. J.). Abgerufen 24. November 2019, von Codecademy website: https://www.codecademy.com/articles/what-is-rest

What is REST – Learn to create timeless REST APIs. (o. J.). Abgerufen 24. November 2019, von https://restfulapi.net/

Woodley, M. S. (2004, April 23). DCMI: Using Dublin Core. Abgerufen 16. November 2019, von https://www.dublincore.org/specifications/dublin-core/usageguide/glossary/

Wrong results from Item Set Finder / Association Rule Learner. (2018, Mai 17). Abgerufen 17. November 2019, von KNIME Community Forum website: https://forum.knime.com/t/wrong-results-from-item-set-finder-association-rule-learner/11026

XML Reader. (o. J.). Abgerufen 28. November 2019, von KNIME Hub website: https://hub.knime.com/knime/extensions/org.knime.features.xml/latest/org.knime.xml.node.reader.XMLReaderNodeFactory

Yang, S. Q., & Li, L. (2015). *Emerging Technologies for Librarians: A Practical Approach to Innovation*. Chandos Publishing.

Zeng, M. L., & Qin, J. (2016). *Metadata* (Second edition, UK edition). London: fp, facet publishing.