XML Entwicklerhandbuch

XSLT - XQuery - MarkLogic

www.github.com/alexdd/Buch

www.tekturcms.de

Version: 1

Tue Aug 13 2024 12:35:26 / en-GB

Diese Publikation wurde mit Tektur CCMS verfasst. Tektur ist ein einfach zu bedienender kollaborativer Editor um DITA[1)] Hier mache ich meine Änderung Inhalte erstellen, als PDF ausgeben und pflegen zu können. Die Eingabe erfolgt dabei per WYSIWYG[2)] mit geführter Benutzerinteraktion. Die Inhalte werden als einzelne Topics verwaltet, die in verschiedenen Maps referenziert werden können; Stichwort: Topic Based Authoring[3)].

- wqdqwd

  qwdqwd
- qwdqwd
- qwdqwd

Sonstige Features: Rechte- und Rollensystem, Versionskontrolle, konfigurierbarer Workflow mit Review & Approval Funktionen. Auf dem Entwicklerblog[4)] kann man sich über den Fortschritt informieren.

1) https://de.wikipedia.org/wiki/Darwin_Information_Typing_Architecture
2) https://de.wikipedia.org/wiki/WYSIWYG
3) https://en.wikipedia.org/wiki/Topic-based_authoring
4) http://www.tekturcms.de

- Das Buch dient in erster Linie als Test für Tektur CCMS. Der Inhalt ist recht schnell entstanden, so dass der Feinschliff an mancher Stelle vielleicht noch fehlt... Trotzdem könnte es für den einen oder anderen XML Entwickler ganz interessant sein. r Linie als Gedächtnisstütze.

  Die Quelltexte sind aus "didaktischen" Gründen an mancher Stelle in deutsch gehalten, was natürlich für ein richtiges Programm nicht soviel Sinn machen würde.

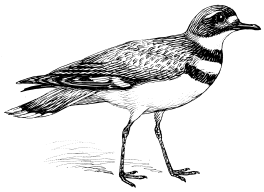
  Die Strichzeichnungen der exotischen Vögel sind von Openclipart[5)]. Meine Lieblingsgrafikdesignerin Caro hat die schönen Girlanden und die Titelgrafik beigesteuert. Vielen Dank dafür :-)

5) https://openclipart.org/

Contents

1 Anwendungsgebiete

Contents



XML, XSLT, **XPATH**, **XSL-FO** und **XQuery** sind Techniken um baumstrukturierte Daten - im Vergleich zu relationalen Daten - aus verschiedenen Quellen ineinander zu überführen, abzuspeichern, zu versenden, darzustellen und auszuwerten können.

Vom Aussehen her sind XML Daten im Prinzip Textdaten. Sie können sehr einfach mit einem Texteditor erstellt werden. Im Gegensatz zu Multimedia-Daten sind keine komplexen Tools, wie z.B. ein Grafikeditor, erforderlich. Auch relationale Daten können in Form von Tabellen, als Excel Tabelle, oder bspw. als kommaseparierte Textdatei, aus einem System ausgespult und weiterverarbeitet werden. XML erlaubt es jedoch die Daten semantisch auszuzeichnen. Das geschieht durch das Klammern semantisch zusammengehöriger Elemente mittels Klammer-Tags und Kategorisierung dieser Informationseinheiten mittels weiterer Properties (Attribute) an diesen Tags. Durch das Verschachteln dieser geklammerten Komponenten entsteht ein Baum, der die Hierarchische Ordnung der Daten widerspiegelt. Diese Baumstrukturen sind maschinell lesbar und die Daten können, bevor sie von einem Versender zu einem Empfänger gehen, mittels eines automatischen Prozesses validiert werden. Dabei können sowohl der Inhalt als auch die Syntax anhand von definierten Regeln (Schemas) genau überprüft werden.

Der XML Standard ist mittlerweile 20 Jahre alt. Zuvor gab es SGML, das zum Beispiel auch nicht abgeschlossene Tags erlaubte. SGML ist die konzeptionelle Basis von HTML.

Der Übergang von SGML zu XML brachte eine längst fällige konzeptionelle Vereinfachung. JSON als defakto Standard bei Webapplikationen vereinfacht den Einsatz von baumstrukturierten Daten noch weiter. JSON ist jedoch nicht so gut maschinenlesbar, und es fehlen noch viele Werkzeuge, wie ausgefeiltere Code Editoren oder Validierungstools.

Folgend eine kurze Erläuterung zu den eingangs erwähnten Schlüsselwörtern:

- **XML** ist das Datenformat. Auf XML arbeiten die anderen Technologien. XML ist immer Input für diese Tools.
- **XSLT** ist eine Programmiersprache. Mit XSLT kann man XML Daten in ein anderes Format transformieren, d.h. Inhalt und Struktur anpassen.
- **XPATH** erlaubt es, bestimmte Knoten in einem XML Dokument über bedingte Pfadausdrücke zu selektieren und dann mittels XSLT zu verändern.
- **XSL-FO** ist eine weitere XML basierte Auszeichnungssprache, die ein XSL-FO Prozessor einlesen kann, um daraus z.B. ein PDF zu generieren.
- **XQuery** ist eine Abfragesprache ähnlich zu SQL, jedoch werden damit nicht relationale Daten abgefragt sondern baumstrukturierte.

1.1 XSLT - die Programmiersprache im XML Bereich

XSLT ist im Bereich XML ganz gross. Ausserhalb kennt man sie allerdings kaum. Im TIOBE Index[6] von 2003 rangierte XSLT einmal auf Platz 60 an letzter Stelle der Liste[7].

6) https://de.wikipedia.org/wiki/TIOBE-Index

Im Bereich XML wäre aber ohne XSLT nicht viel möglich. Es gibt einige exotische Anwendungsgebiete in denen XML effizient mit LISP Dialekten verarbeitet wird, bspw. die Verarbeitung von - nach XML konvertierten - EDI X12[8)] Nachrichten.

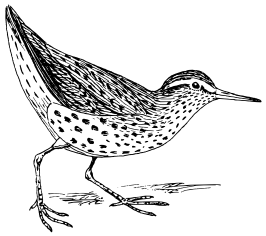
SGML, der Vorreiter von XML, hat sich als S1000D Standard[9)] im Bereich Luftfahrt wacker gehalten. Hier wird teilweise noch mit proprietären Programmiersprachen, wie **Metamorphosis**, gearbeitet.

Alternativen zu XSLT finden sich im entsprechendem Wikipedia-Artikel[10)].

Wir konzentrieren uns im Rahmen dieser Lektüre auf XSLT und XSL Stylesheets - damit wird i.A. die Verbindung von XSLT, XSL-FO und XPath gemeint, um damit XML Daten in andere Formate zu überführen, bspw. PDF.

Beispielsweise werden die Autohandbücher führender Hersteller mittels XSL gesetzt, deren Eingabedaten aufbereitet und zur Weiterverarbeitung transformiert.

Ein paar interessante Stichpunkte:



- XSLT hat gerade noch den Status "Programmiersprache", weil man damit eine Turing Maschine[11)] programmieren kann.
- Mit HTML oder einer Templater Sprache (z.B. JSP) würde das nicht funktionieren.
- XSLT ist keine imperative Sprache, d.h es werden keine Anweisungen der Reihe nach abgearbeitet, sondern eine deklarative Sprache, d.h für jedes Ereignis - besser gesagt - für jeden durchlaufenen DOM Knoten wird eine gefundene und vom Programmierer deklarierte Regel angewendet.
- Ausserdem gibt es funktionale Anteile, um bspw. die deklarierten Regeln rekursiv anwenden zu können. So können auch größere Programme sinnvoll strukturiert werden.
- XSLT wird oft mit XSL gleichgesetzt. Aber XSL[12)] ist mehr:
  - Zum einen kommt noch XPath hinzu: XPath erlaubt komplizierte Berechnungen und Selektionen auf den DOM[13)] Knoten eines XML Dokuments.
  - Zum anderen ist auch XSL-FO Bestandteil der XSL Spezifikation[14)]. XSL-FO Tags sind Anweisungen für einen XSL-FO Prozessor, der aus einem XSL-FO Dokument ein PDF Dokument generiert. Es sind auch andere Ausgabe-Formate, wie bspw. RTF möglich.

1.2 Aktuelle und vergangene Anwendungen

Wie auch bei anderen Programmiersprachen, hat es einige Zeit gedauert bis sich für XSLT der optimale Anwendungsbereich herauskristallisiert hat. Auch bei XSLT war ursprünglich das Internet und insbesondere Webseitenprogrammierung die treibende Kraft, da mittels XSLT besonders gut Inhalt und Semantik getrennt werden konnte.

Relativ schnell hat sich aber **CSS** in Verbindung mit **JavaScript** als Standardlösung für diesen Zeck bewährt. **XSLT** ist inzwischen Platzhirsch im Bereich **Technische Dokumentation,** und hier auch wohl unschlagbar.

7) https://bit.ly/2ARgKCJ
8) https://en.wikipedia.org/wiki/ASC_X12
9) https://en.wikipedia.org/wiki/S1000D
10) https://de.wikipedia.org/wiki/XSL_Transformation
11) http://www.unidex.com/turing/utm.htm
12) https://de.wikipedia.org/wiki/XSL_Transformation
13) https://de.wikipedia.org/wiki/Document_Object_Model
14) https://www.w3.org/TR/xsl/

XML Webseiten

Einen XSLT Prozessor hat jeder Browser eingebaut. Es war mal sehr populär, Webseiten als XML auszuliefern und mittels XSLT zu layouten. U.a. wegen des exzessiven Einsatzes von JavaScript (auch inline), hat sich diese Idee nie vollständig durchgesetzt. Schliesslich wurde **XHTML** spezifiziert und jetzt gibt es **HTML5**.

Betrachten wir das folgende XML Beispiel:

```
<?xml version="1.0" encoding="UTF-8"?>
<document>
<title>Das ultimative Zwei-Kapitel Dokument</title>
  <chapter>
    <title>Kapitel 1</title>
    <intro>In Kapitel 1 wird kurz gesagt was Sache ist.</intro>
    <content>Um es kurz zu machen, wie der Hase läuft stehtin Kapitel 2.</content>
  </chapter>
  <chapter>
    <title>Kapitel 2</title>
    <intro>Hier wird erklärt, wie der Hase läuft.</intro>
    <content>Im Prinzip ist es ganz einfach.</content>
  </chapter>
</document>
```

Ohne XSLT Stylesheet Zuweisung wird der Browser eine Datei mit diesem Inhalt als eingerücktes XML anzeigen - oder die Tags einfach ignorieren und den Textinhalt in einer Zeile darstellen. Fügt man eine Processing Instruction[15] am Anfang ein, wird ein XSLT Stylesheet vom Browser herangezogen, und vor der Darstellung im Browser wird die so deklarierte XML Transformation ausgeführt:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="formatiermich.xsl" ?>
<document>
  <title>Das ultimative Zwei-Kapitel Dokument</title>
  <chapter>
[...]
```

Das XML kann nun im Browser geöffnet werden. Alles wird schön formatiert angezeigt...

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform">

  <xsl:template match="/">
    <html>
      <xsl:apply-templates/>
    </html>
  </xsl:template>

  <xsl:template match="document">
  <body>
      <xsl:apply-templates/>
    </body>
  </xsl:template>

  <xsl:template match="document/title">
    <h1>
      <xsl:apply-templates/>
    </h1>
  </xsl:template>
```

15) https://de.wikipedia.org/wiki/Verarbeitungsanweisung

```
  <xsl:template match="chapter">
    <div class="chapter">
      <xsl:apply-templates/>
    </div>
  </xsl:template>

  <xsl:template match="chapter/title">
    <h2>
      <xsl:apply-templates/>
    </h2>
  </xsl:template>

  <xsl:template match="chapter/intro">
    <div class="intro">
      <i><xsl:apply-templates/></i>
    </div>
  </xsl:template>

  <xsl:template match="chapter/content">
    <p><xsl:apply-templates/></p>
  </xsl:template>
</xsl:stylesheet>
```

Die **Processing Instruction** hat keinen Einfluss auf den XML Inhalt und wird in einer anderen Eingabeverarbeitung nicht herangezogen.

Serverseitige Konvertierung

Auch eine serverseitige Konvertierung ist gebräuchlich. Ein Beispiel aus vergangenen Tagen - WAP-Seiten[16] für unterschiedliche Handy-Modelle.

Früher hatten die Handys sehr unterschiedliche Displaygrößen. Handybrowser konnten nicht ausreichend JavaScript und die Skalierung der WAP-Seite für das jeweilige Handy passierte nicht im Handy, sondern vor der Auslieferung auf der Serverseite. Dazu wurde eine XML Quelle mittels verschiedener XSLT Stylesheets in unterschiedliche WML WAP Repräsentationen transformiert.

So würde das Zwei-Kapitel Beispiel von oben im WML Format aussehen (recht einfach gehalten):

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE wml PUBLIC "-//WAPFORUM//DTD WML 1.1//EN" "http://www.wap.org/DTD/wml_1.1.xml">
<wml>
  <head>
    <meta name="title" content="Das ultimative Zwei-Kapitel Dokument"/>
  </head>
  <card id="chapter1" title="Kapitel 1">
    <p><i>In Kapitel 1 wird kurz gesagt was Sache ist.</i></p>
    <p>Um es kurz zu machen, wie der Hase läuft stehtin Kapitel 2.</p>
  </card>
  <card id="chapter2" title="Kapitel 2">
    <p><i>Hier wird erklärt, wie der Hase läuft.</i></p>
    <p>Im Prinzip ist es ganz einfach.</p>
  </card>
</wml>
```

Eine XSLT Transformation, die die XML Daten von oben in diese **WML** Darstellung überführt, könnte z.B. so implementiert werden:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="1.0">

  <xsl:output
```

16) https://de.wikipedia.org/wiki/Wireless_Application_Protocol

```
    doctype-public="-//WAPFORUM//DTD WML 1.2//EN"
    doctype-system="http://www.wapforum.org/DTD/wml12.dtd"
    indent="yes"/>

  <xsl:template match="document">
    <wml>
      <xsl:apply-templates/>
    </wml>
  </xsl:template>

  <xsl:template match="document/title">
    <head>
      <meta name="title">
        <xsl:attribute name="content">
          <xsl:value-of select="."/>
        </xsl:attribute>
      </meta>
    </head>
  </xsl:template>

  <xsl:template match="chapter">
    <card id="{concat('chapter',count(preceding-sibling::chapter)+1)}">
      <xsl:attribute name="title">
        <xsl:value-of select="title"/>
      </xsl:attribute>
      <xsl:apply-templates select="*[not(self::title)]"/>
    </card>
  </xsl:template>

  <xsl:template match="node()|@*">
    <xsl:copy>
      <xsl:apply-templates select="node()|@*"/>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="processing-instruction()"/>

  <xsl:template match="intro">
    <p><i><xsl:apply-templates/></i></p>
  </xsl:template>

  <xsl:template match="content">
    <p><xsl:apply-templates/></p>
  </xsl:template>
</xsl:stylesheet>
```

Multiple Ausgabeformate

Aus einer XML Quelle können auch leicht weitere Format erzeugt werden, bspw. EPUB[17] Das ist das Standardformat für eBooks. Neben Tags zur Formatierung für den Content, gibt es bspw. auch Anweisungen zum Erzeugen des Inhaltsverzeichnisses oder anderer Navigationsstrukturen.

Weitere gängige Formate sind neben dem oben veralteten WML Format, elektronische Ausgabe-Formate wie: CHM[18], EclipseHelp[19], JavaHelp[20], ..., Print-Ausgabe Formate, wie PDF oder Adobe Framemaker[21], oder XML Standard Austauschformate, wie **DITA**, **S1000D**, PI-MOD[22], JATS[23] oder TEI[24] .

17) https://de.wikipedia.org/wiki/EPUB
18) https://de.wikipedia.org/wiki/CHM_(Dateiformat
19) https://www.ibm.com/developerworks/library/os-echelp/index.html
20) https://en.wikipedia.org/wiki/JavaHelp
21) https://de.wikipedia.org/wiki/FrameMaker
22) https://www.i4icm.de/forschungstransfer/pi-mod/
23) https://de.wikipedia.org/wiki/Journal_Article_Tag_Suite
24) https://de.wikipedia.org/wiki/Text_Encoding_Initiative

1 Anwendungsgebiete

➢ Aktuelle und vergangene Anwendungen

Menschenlesbare Ausgabe

Kryptische XML Log-, Daten- oder Konfigurationsfiles können leicht mit XSLT "menschenlesbar" formatiert werden. Ein Arbeitskollege im neuen Job kam kürzlich auf mich zu, ob ich um eine Möglichkeit wüsste, wie man sein kryptisches Datenfile für einen Übersetzungsdienst formatieren könnte:

```
<?xml version="1.0" encoding="UTF-8"?>?<?xml-stylesheet type="text/xsl" href="de.xsl"?>
<jcr:root xmlns:sling="http://sling.apache.org/jcr/sling/1.0"
  xmlns:jcr="http://www.jcp.org/jcr/1.0"
  xmlns:mix="http://www.jcp.org/jcr/mix/1.0"
  xmlns:nt="http://www.jcp.org/jcr/nt/1.0"
  jcr:language="de"
  jcr:mixinTypes="[mix:language]"
  jcr:primaryType="sling:Folder">
<b_manual
  jcr:primaryType="sling:MessageEntry"
  sling:message="Bedienungsanleitung"/>
  <b_warning
    jcr:primaryType="sling:MessageEntry"
    sling:message="Warnung"/>
  <b_danger
    jcr:primaryType="sling:MessageEntry"
    sling:message="Vorsicht"/>
  <b_note
    jcr:primaryType="sling:MessageEntry"
    sling:message="Notiz"/>
  <b_notice
    jcr:primaryType="sling:MessageEntry"
    sling:message="Hinweis"/>
  [...]
```

Mit einem eingehängten XSLT Stylesheet `de.xsl` wird so ein Datenfile als Tabelle formatiert:

```
<?xml version="1.0" encoding="iso-8859-1"?>
<xsl:stylesheet version="1.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:jcr="http://www.jcp.org/jcr/1.0"
  xmlns:sling="http://sling.apache.org/jcr/sling/1.0">

  <xsl:template match="jcr:root">
    <html>
      <table border="1" cellpadding="5" cellspacing="5">
        <xsl:apply-templates/>
      </table>
    </html>
  </xsl:template>

  <xsl:template match="*">
    <tr>
      <td>
        <xsl:value-of select="concat(count(preceding::*[@sling:message]) + 1,'.')"/>
      </td>
      <td>
        <xsl:value-of select="name()"/>
      </td>
      <td contenteditable="true">
        <xsl:value-of select="@sling:message"/>
      </td>
    </tr>
  </xsl:template>
</xsl:stylesheet>
```

Hängt man an dieses Beispiel noch ein bisschen JavaScript Logik und macht die Felder für die Übersetzungen mittels des **HTML5** `contenteditable` Attributs editierbar, dann bräuchte man nur noch eine Rücktransformation HTML nach XML und hätte schon einen kleinen XML Editor gebaut. So funktioniert auch der Editor in Tektur.

Diagramme darstellen

Nachdem eine **SVG** Grafik im **XML** Format vorliegt, kann diese auch direkt mittels XSLT erzeugt werden. Über das HTML5 `<svg>` Element kann so eine Grafik inline in das - ebenfalls durch das XSLT - generierte HTML Dokument eingebunden werden.

Betrachten wir unser Beispiel von oben, erweitert um drei neue `<block>` Elemente:

```
<?xml version="1.0" encoding="UTF-8"?>
<?xml-stylesheet type="text/xsl" href="chart.xsl" ?>
<document>
  <title>Das ultimative Zwei-Kapitel Dokument</title>
  <chapter>
    <title>Kapitel 1</title>
    <intro>In Kapitel 1 wird kurz gesagt was Sache ist.</intro>
    <content>Um es kurz zu machen, wie der Hase läuft stehtin Kapitel 2.</content>
  </chapter>
  <chapter>
    <title>Kapitel 2</title>
    <intro>Hier wird erklärt, wie der Hase läuft.</intro>
    <content>Im Prinzip ist es ganz einfach. Betrachten wir doch drei gelbe Blöcke:
    </content>
    <block/>
    <block/>
    <block/>
  </chapter>
</document>
```

Wenn wir das XSLT Stylesheet noch um eine Regel für das neue `<block>` Element ergänzen, so wie hier:

```
<xsl:template match="block">
   <svg style="background-color:yellow" width="30" height="30"
     xmlns:xlink="http://www.w3.org/1999/xlink"
     xmlns="http://www.w3.org/2000/svg"/>
  <br/>
  <br/>
</xsl:template>
```

Dann erhalten wir drei schön formatierte gelbe SVG Blöcke ...

Weiterführende Links:

- Client-side image generation with SVG and XSLT[25)]
- Knotentyp Visualisierung im Apache Jack Rabbit Projekt[26)]

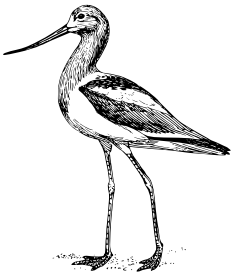
1.3 Professionelle XML Verarbeitung

Vom Single-Source Publishing bis zur Generierung von Java Code aus Klassendiagrammen. XSLT ist in vielen großen Softwareprojekten vertreten. Heute liefert eine Suche nach `<xsl:stylesheet` bspw. 14.868.501 Treffer.[27)]

25) http://surguy.net/articles/client-side-svg.xml
26) http://jackrabbit.apache.org/jcr/node-type-visualization.html
27) https://github.com/search?q=xsl%3Astylesheet&type=Code

Single Source Publishing

Aus einer Quelle werden viele Ausgabe Formate generiert. Gängige Formate in der Technischen Dokumentation sind elektronische Ausgabe-Formate wie: **CHM**, **Eclipse-Help**, **JavaHelp**, **ePub**, ..., Print-Ausgabe Formate, wie **PDF** oder **Adobe Framemaker**, oder **XML** Standard Austauschformate, wie **DITA**, **S1000D**, **PI-MOD** oder **TEI** .



Vorteile:

- Bei einer Änderung in der XML Quelle werden auch automatisch alle anschließenden Formate aktualisiert.
- Strikte Trennung von Content/Semantik und Layout/Design.
- Auf der XML Quelle sind XML Features möglich, wie:
  - **Modularisierung** erlaubt die fein-granulare Wiederverwendung von Content-Bausteinen, sowie das Verlinken, Filtern, Suchen und Exportieren derselben.
  - **Generalisierung**

    ist ein DITA Konzept, welches die Wiederverendung von angepassten Topics in anderen DITA Systemen ermöglicht.
  - **Gültigkeiten** erlauben die bedingte Anwendung von Content-Bestandteilen auf Satz- und Wort-Ebene.
  - **Versionierung** und Diffing - Vergleich von Änderungen zwischen Versionen.
  - **Intelligente Querverweise** : Ein Link zwischen einzelnen XML Topics bleibt versionstreu.
  - **Automatischer Satz**, inkl. Zusamenhalte- und Trennregeln für Seiten, Absätze und Blöcke (Listen, Tabellen, etc).
- Veraltete Formate können ausgetauscht werden, ohne dass der Content geändert werden muss oder verlorengeht.
- Die XML Quelle kann ohne Aufbereitung in anderen Systemen wiederverwendet werden.
- Es gibt weit verbreitete **Standards** zur Struktur der XML Quelle.
- Nur das XML wird in der Datenhaltung persistiert.
- Es gibt spezialisierte XML Datenbanken on page 103, die besonders gut auf Baumstrukturen arbeiten. (Dokumente sind per se baum-strukturiert und sind eigentlich für eine relationale Datenbank ungeeignet)

Die Redaktionssysteme führender Hersteller in Deutschland haben XML unter der Haube und setzen auf die Single-Source Strategie.

Code Generierung

Nachdem man bei XSLT im Format der Ausgabe frei ist, kann auch direkt Plain-Text mit XSLT Regeln generiert werden. Daher liegt es nahe sich jegliche Form von Quelltext aus einer XML Repräsentation erzeugen zu lassen.

Beispielsweise speichern gängige **CASE Tools** (Computer Aided Software Engineering) UML Diagramme im XML Format ab, so z.B. ArgoUML[28].

Diese Klassendiagramme lassen sich mittels XSLT direkt in Java-Code transformieren, wie z.B. in einem kleinen Open Source Projekt (aus vergangenen Tagen) : Butterfly Code Generator[29]

Es gibt auch einen schönen Artikel dazu im Java World Journal[30].

Migrationen

Für jede erdenkliche Art der Migration eines XML Datenbestands oder eines Datenbank-Dumps / -Exports im XML Format, zwischen Produktversionen oder zwischen Dienstleister- und Dienstnutzer-Systemen, bietet sich XSLT zur Transformation an.

Dabei ist zu beachten, dass XSLT besonders schnell und gut auf verschachtelten Strukturen arbeitet. Entartet ein Baum zur Liste und/oder sind nur geringe Strukturanpassungen notwendig, wird man sich mit einem schnellen SAX Parser leichter tun.

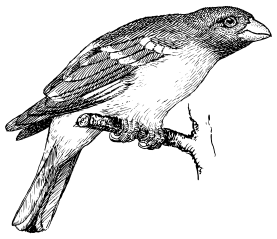
Mittels der XSLT 3.0 Streaming Option können auch sehr große XML Quellen (Big Data) verarbeitet werden. Saxon bietet bspw. diese Streaming Option[31].

1.4 Technische Dokumentation

95% aller technischen Dokumente werden vermtl. mit Micro$oft Word produziert. Es gibt jedoch einige Use Cases in denen ein Standard Word Prozessor nicht mehr ausreicht.

Use Cases

Maschinenbauer



Ein Maschinenbauhersteller verwendet bestimmte Bauteile in mehreren Maschinen. Die Dokumentation dieser Parts soll mit wenig Aufwand in verschiedenen Büchern wiederverwendet werden. Copy 'n Paste kommt nicht in Frage, da sich wegen des schnellen technischen Fortschritts, die Dokumentation häufig ändert. Zudem läuft auf etwas älteren Maschinenmodellen noch eine alte Version von Micro$oft Windows, so dass ein Service-Techniker an der Maschine auf eine Dokumentation im veralteten Windows Hildeformat CHM angewiesen ist.

KFZ Hersteller

Ein KFZ Hersteller hat verschiedene Automodelle im Angebot. Da sich die Fahrzeuge in über hundert Ländern verkaufen lassen, sind Handbücher in vielen Sprachen verfügbar. Es enstehen enorme Übersetzungskosten, selbst für Teile, die in allen Büchern denselben Inhalt tragen, wie bspw. bestimmte Warnhinweise.

Kommunikationstechnik

Ein Hersteller in der Kommunikations- und Signalerfassungstechnik will die Erstellung von Datenblättern automatisieren, indem direkt Messergebnisse aus einer Datenbank in die Dokumentation wandern. Dadurch werden die Kosten bzgl. einer manuellen

28) http://argouml.tigris.org
29) http://butterflycode.sourceforge.net
30) https://www.javaworld.com/article/2073998/java-web-development/generate-javabean-classes-dynamically-with-xslt.html
31) http://www.saxonica.com/html/documentation/sourcedocs/streaming/

Bedatung minimiert. Gesucht ist ein automatisierter Prozess, der die bereitgestellten Daten zur Dokumentation hinzufügt.

Fluggesellschaft

Eine Fluggesellschaft stellt für ihre Piloten eine eigene Dokumentation zu ihren Flugzeugtypen bereit. Da die Piloten bei der Erstellung der Doku als sogenannte "Subject Matter Experts" mitwirken, genügt es nicht nur ein Handbuch in Papierform zu produzieren, sondern auch ein interaktives Portal ist notwendig, mit dem die Piloten die Dokumentation Korrekturlesen und Freigeben können. Schliesslich brauchen sie auch noch eine Version der Doku auf dem Tablet, um bspw. Checklisten vor dem Start interaktiv abhaken zu können - diese Funktionalität rangiert gewöhnlich unter der Bezeichnung "Electronic Flightbag - EFB" .

Konzepte

Um die zuvor genannten Use Cases realisieren zu können, wurden im Bereich Technische Dokumentation Konzepte entwickelt, die sich stets - unter Einbezug der aktuellen technischen Möglichkeiten - weiterentwickeln. Einige dieser Konzepte sind im Kapitel Anwendungsgebiete on page 7 schon kurz genannt. Im Bezug auf die Use Cases oben sind die folgenden interessant:

Modularisierung und Wiederverwendung

Durch eine feingranulare Modularisierung des Content existiert jede Informationseinheit im System nur einmal, und kann in verschiedenen Publikationen referenziert werden. Die Auflösung dieser Referenzen geschieht zum Zeitpunkt der Auslieferung des Ausgabeformats, d.h. bei der Bestückung des Webservers mit einer Online-Dokumentation oder beim Erzeugen von XSL-FO als Vorformat für eine Print-Publikation mittels PDF.

Single Sourcing

Aus einer Quelle werden mehrere Ausgabeformate erzeugt. Dadurch wird Redundanz vermieden und alle Publikationen können so stets aktuell gehalten werden. Wenn sich die Quelle ändert, wird automatisch auch das Ziel aktualisiert. Gängige Ausgabeformate sind PDF, Online-Hilfe, Online-Portale, eBook Formate, usw. Auch veraltete Legacy Formate, wie Windows Hilfe CHM, können bei Bedarf über eine neu hinzugefügte Ausgabestrecke realisiert werden ohne den Kern des Systems zu belasten.

Variantensteuerung

Über konditionale Bedingungen *"Gültigkeiten"* auf Kapitel- und Elementebene kann die Ausgabe für verschiedene Produktvarianten feingranular gesteuert werden. Dabei werden z.B. die Varianten in einer (Entscheidungs-) baumstruktur unabhängig von der Publikation verwaltet. Erst beim Publizieren wird entschieden, welche Grafiken und Satzbausteine herangezogen werden.

Übersetzungsmanagement

Die Übersetzung einer Quellsprache in eine Zielsprache wird gemeinsam mit der Quellinformationseinheit verwaltet. So kann auch auf Seiten der Redakteure eine Versionshistorie bzgl. der Übersetzungen gepflegt werden. Die Abhängigkeit zum Übersetzungsdienstleister wird dadurch minimiert. Einmal getätigte Übersetzungen können wiederverwendet werden und belasten das Budget nicht.

Automatischer Satz mit XML Technologie

XML ist gängiges Datenformat in der Industrie. XML Daten gelangen aus unterschiedlichen Quellen und über verschiedene Wege, wie Datenbanken, Webservices, mittels BPM Prozesse, Ablageverzeichnisse, manueller Bedatung, usw. in eine Publikationsinstanz.

Diese wird als Single Source Quelle in die Ausgabeformate transformiert. Natürlich verlangen diese automatischen Eingabeprozesse auch eine automatische Ausgabe. Für Online-Formate ist das weniger tragisch, aber für Print-Formate zählt ein guter

Umbruch auf Paragraph-, Spalten-, Tabellenzeilen- und Seitenebene. Auch Grafiken und Tabellen müssen gut umbrechen, bspw. soll der Untertitel immer an der Grafik gehalten werden und Tabellenzeilen sollten, auch wie Paragraphen, keine Hurenkinder und Schusterjungen[32] erzeugen.

Moderne End-User Applikationen

Schlussendlich liefert Word keine Datenbasis für moderne Applikationen, wie interaktive Portale oder Smartphone- bzw. Tabletapplikationen. Gegenstand aktueller Forschung sind VR und AR, bspw. sollte es genügen eine bestimmte Maschine im Fokus seines Smartphones zu halten, um relevante Technische Dokumentation zu übertragen. Auch werden neue Metadatenkonzepte erforscht, wie die Modellierung der Beziehungen zwischen einzelnen Informationseinheiten über RDF Tripel.

```
Bsp.: Bauteil A hat Funktion X in Maschine Q aber Bauteil A hat Funktion Y in Maschine W.
Zeige mir alle Funktionen von Bauteil A.
```

Spezielle Content Delivery Portale spannen dazu einen Beziehungsgraphen auf und ermöglichen das schnelle Bereitstellen der gesuchten Information. Federführend ist hier der iiRDS Standard[33].

Werkzeuge

Um die genannten Use Cases umfassend zu erschlagen sind spezielle Content Management Systeme erforderlich, sogenannte Component Content Management Systeme[34]. Diese Systeme existieren größtenteils seit den 90er Jahren, laufen auf dem Desktop und erfordern enorme Betriebs- und Einführungskosten, die sich nur größere Konzerne und Unternehmungen leisten. Tektur CCMS[35] versucht hier mit einer modernen, webbasierten und kostengünstigen Lösung in die Bresche zu springen.

32) https://de.wikipedia.org/wiki/Hurenkind_und_Schusterjunge
33) https://iirds.org/
34) https://en.wikipedia.org/wiki/Component_content_management_system
35) http://www.tekturcms.de

2 Wichtige Konzepte

Contents

XSLT und **XQuery** erlauben es Probleme auf viele verschiedene Arten zu lösen. Sicherlich wird jeder Programmierer im Laufe der Zeit seinen eigenen Stil entwickeln.
Das kommt nicht zuletzt daher, dass man als XSLT Entwickler in vielen Firmen eine Expertenrolle einnimmt.

Umso wichtiger ist es, sich an allgemeine Konzepte, Muster und Best Practices zu halten, um einen schwer wartbaren Wildwuchs zu vermeiden.

Auf den folgenden Seiten wird versucht einige dieser Konzepte zusammenzutragen und mit eigenen Erfahrungen und Ideen zu kombinieren.

Es wird weder der Anspruch auf Vollständigkeit noch auf 100%-ige Korrektheit dieser Informationen erhoben. Das Kapitel soll vielmehr als Denkanstoß - mit hoffentlich einigen verwertbaren Ideen - dienen.

2.1 Push vs. Pull Stylesheets

XSLT ist eine ereignisgesteuerte, regelbasierte Umgebung zur Konvertierung von XML Daten. Gerade der Vorteil des regelbasierten Ansatzes ist vielen Entwicklern nicht bewusst, und es entsteht Quellcode der aussieht, wie mit XPath angereicherter **PHP** Code.

Wieso nimmt man dann überhaupt XSLT, wenn man keine Template-Match Regeln verwendet, oder nur spärlich verwendet?

Um diesen Umstand aufzuklären ist ein bisschen Theorie notwendig:

⊳ Push vs. Pull Stylesheets

Beim "Pull" werden Elemente in der Quellinstanz selektiert und an einer passenden Stelle in der Zielinstanz eingefügt. Diese Vorgehensweise ist vergleichbar mit derer von Template-Engines, wie **JSP** oder **ASP**. Das kann in mehreren Stufen erfolgen, bis schrittweise die Quellinstanz in die finale Zielinstanz überführt wurde.

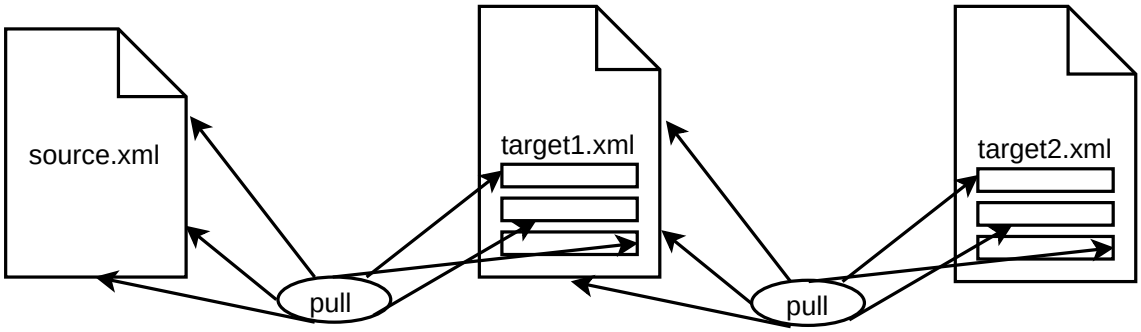


figure 1: Pull Stylesheet

Beim "Push" werden die Quelldaten schrittweise in die Zieldaten konvertiert. Diese Vorgehensweise kann explorativ erfolgen und beim Transformieren in einen Zwischenschritt entstehen Erkenntnisse, die bei der Weiterverarbeitung nützlich sind. Merke:XSLT steht für eXtensible Stylesheet Transformation.

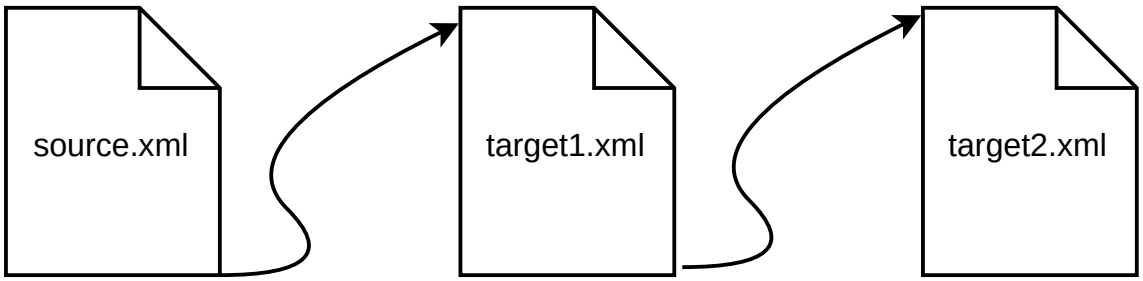


figure 2: Push Stylesheet

Das bisher Gesagte verdeutlicht zwar den "Pull" Ansatz, was genau aber ge"pusht" wird, ist vermutlich noch unklar. Betrachten wir XML in der Baumdarstellung.

Der XSLT Prozessor unternimmt einen Tiefensuchlauf und überprüft bei jedem Knoten den er betritt, ob in seiner Regelbasis eine Regel existiert, die auf diesen Knoten "matched". Dabei gibt es drei grundsätzliche Möglichkeiten, wie die Knoten des Quellbaums in den Zielbaum kopiert - oder eben nicht kopiert - werden können.

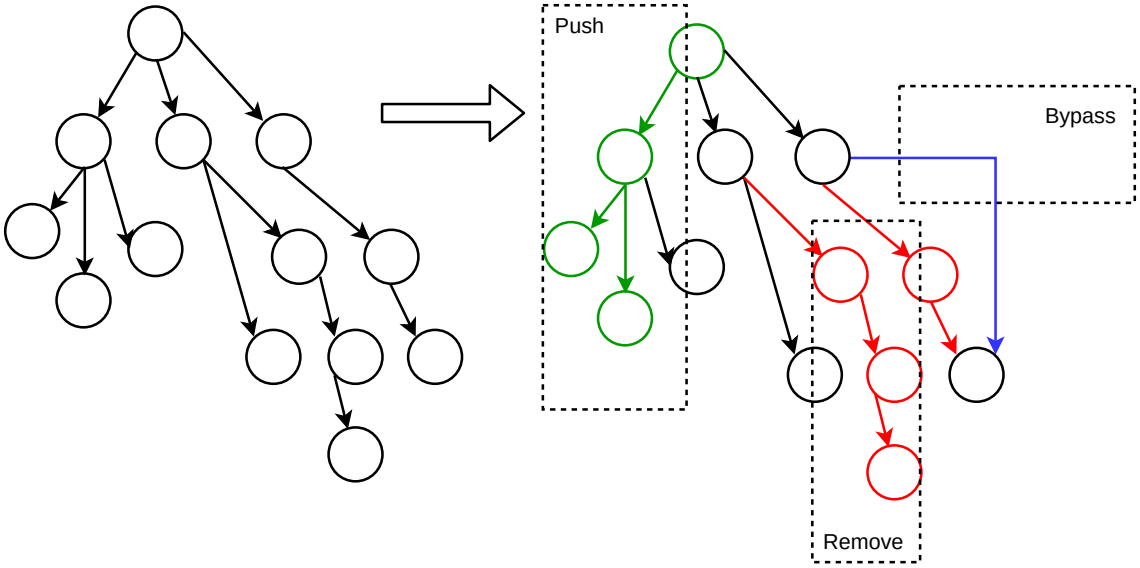


figure 3: Transformation des Quellbaus in den Zielbaum

Pull-Stylesheets werden gewöhlich mit `xsl:for-each` Loops programmiert. Dieser Ansatz ist meiner Ansicht nach gebräuchlich, wenn keine großen **DTD** Änderungen zu erwarten sind, der XML Baum flach strukturiert ist und die Anforderungen an die Konvertierung relativ einfach sind, bspw. beim Auswerten / Konvertieren von Konfigurationsdateien. In allen anderen Fällen sind Push-Stylesheets vorzuziehen, d.h. möglichst wenig `xsl:for-each` loops und möglichst viele Template-Match Regeln.

2.2 Eindeutigkeit der Regelbasis

Die Regelbasis von XSLT kann wahrscheinlich unendlich viele Regeln aufnehmen, wenn man unsere Limitierung bzgl. der Hardware nicht beachtet. Für die Vollständigkeit, Eindeutigkeit und Konsistenz der Regelbasis ist der Programmierer aber selbst verantwortlich.

Um die Eindeutigkeit der Regeln zu gewährleisten, gibt es verschiedene Mechanismen.

Reihenfolge der Match-Regeln

Im Normalfall sollte auf einen bestimmten Knoten in einem bestimmten Szenario genau eine Regel matchen. Falls es einen Konflikt gibt, wird zumindest bei **Saxon** diejenige Regel herangezogen, die im Stylesheet zuletzt deklariert wurde.

Diesen Umstand zu kennen, ist genau dann wichtig, wenn man einen bestehenden Stylesheet-Code übernehmen muss. Getreu dem Motto "Never change a running system" sollte man die Sache diesbzgl. sehr behutsam aufräumen.

Präzedenz der Auswertung

Match-Regeln werden gemäß ihrer Spezifität sortiert und diejenige, die auf einem Knoten in einem bestimmten Szenario am besten zutrifft, wird zur Auswertung herangezogen. Grds. werden die Regeln anhand folgender Kriterien sortiert:

1. Importierte Template Regeln haben immer eine niedrigere Priorität als die Regeln des importierenden Stylesheets.
2. Templates mit einem höheren Priority Attribut haben Vorrang.

⮞ Eindeutigkeit der Regelbasis

3. Templates ohne Priorität bekommen automatisch eine Default-Priorität. Die höchste Default-Priorität ist `0.5` .
4. Diese Default Priorität errechnet sich anhand der Bedingungen oder Wildcards, die an einen Match-Regel geknüpft sind:
   - Wenn mehrere Templates matchen, dann wird das am meisten spezifische zur Auswertung herangezogen.
   - Das am meisten spezifische Template wird anhand der Prioritäten berechnet.
   - Einfache Elementnamen (z.B. "para") haben Prio `0` .
   - Wildcards (z.B. `*, @*`) haben Priorität `-0.25`
   - Knoten-Tests für andere Knoten (e.g. `comment(), node()` , etc.) haben Priorität `-0.5`
   - In allen anderen Fällen ist die Prio `0.5` .

   Beispiele:

   - `para -> 0`
   - `h:* -> -0.25`
   - `* -> -0.25`
   - `node() -> -0.25`
   - `contents/para -> 0.5`
   - `contents/* -> 0.5`
5. Mit einer Kommandozeilen-Option kann bei **Saxon** festgelegt werden, dass die Transformation abbricht, sobald es einen Konflikt bei der Regelauswertung gibt.

Import und Default-Regel

Wie in der obigen Sektion unter Punkt 1. angegeben, haben alle Regeln in einem importierten Stylesheet eine geringere Priorität als im importierenden Stylesheet. Diesen Umstand kann man sich zunutze machen, um eine Default-Regel einzubinden, bspw:

```
<xsl:template match="*" mode="#all"/>
```

Da diese Regel sich in einem importierten Stylesheet befindet, hat sie geringere Priorität als alle anderen Regeln und greift nur dann, wenn für einen betretenen Knoten keine anderen Match-Regeln definiert sind.

Das ist z.B. praktisch, um nicht "gehandelte" Element zu identifizieren - dazu wäre die obige Regel nicht leer, sondern würde bspw. einen gelb markierten Warntext direkt in das Ausgabeformat schreiben.

Eine leere Default-Regel ist dagegen gut, wenn bspw. in einer **XML-2-XML Migration** automatisch Knoten im XML Baum abgetrennt werden sollen...

Prioritäten

Alle Match-Regeln werden mit einer Priorität ausgestattet. Der Stylesheet-Entwickler hat die Möglichkeit diese Priorität zu überschreiben. Dazu wird das Attribut `@priority` an der Match-Regel verwendet. Ein Use-Case für die Prioritäten wäre bspw. folgendes Filter-Szenario

Beispiel Seminarverwaltung

- Die Eingabeinstanz soll in einer Vorprozessierung gefiltert werden.
- Dabei sollen Seminar-Elemente markiert werden, die nicht besonderen Bedingungen entsprechen:
  - Das Seminar-Element hat ein Feld "Ende-Datum" das abgelaufen ist.

⮞ Eindeutigkeit der Regelbasis

  - Am Seminar-Element sind mehrere Dozenten angestellt, obwohl das Seminar-Element vom Typ "Single" ist.
  - Einem Seminar-Element ist kein Dozent zugeordnet.

- Sicherlich kann es Seminar-Elemente geben, die alle drei Bedingungen erfüllen. Um das Error-Log aber nicht zu überfüllen, sollen die Filter nach ihren Prioritäten ausgeführt werden.

In XSLT Templates überführt, könnte diese Anforderung so umgesetzt werden:

```
<xsl:template match="Seminar[Ende-Datum/xs:date(.) le current-date()]"
              priority="30" mode="filter-seminare">
  <xsl:element name="Filtered-Seminar" namespace="{namespace-uri()}">
    <xsl:attribute name="reason">termed-seminar</xsl:attribute>
    <xsl:apply-templates select="node()|@*" mode="filter-seminare"/>
  </xsl:element>
</xsl:template>

<xsl:template match="Seminar[Type eq 'SINGLE' and count(dozenten/dozent) gt 1]"
              priority="20" mode="filter-seminare">
  <xsl:element name="filtered-Seminar" namespace="{namespace-uri()}">
    <xsl:attribute name="reason">dozenten-count</xsl:attribute>
    <xsl:apply-templates select="node()|@*" mode="filter-seminare"/>
  </xsl:element>
</xsl:template>

<xsl:template match="Seminar[not(dozenten/dozent)]" mode="filter-seminare">
  <xsl:element name="filtered-Seminar" namespace="{namespace-uri()}">
    <xsl:attribute name="reason">dozenten-missing</xsl:attribute>
    <xsl:apply-templates select="node()|@*" mode="filter-seminare"/>
  </xsl:element>
</xsl:template>
```

Neben des Einsatzes des `@priority` Attributs und des nachfolgend beschriebenen `@mode` Attributs ist sicherlich auch noch interessant, dass die gefilterten Seminar-Elemente hier nicht gelöscht werden, sondern umbenannt werden. Auf diese Weise können sie in einem nachfolgenden Transformationsschritt (Stichwort: Vortransformationen on page 50) weiterbehandelt werden, stören aber in der regulären Verarbeitung nicht.

Modus Attribute

An allen Templates hat man die Möglichkeit einen selbst deklarierten Modus anzugeben. Wenn dann der XSLT Prozessor in eine bestimmte Richtung gepusht, vgl. Push vs. Pull Stylesheets , wird, werden nur diejenigen Regeln zur Auswertung herangezogen, die im selben Modus sind, wie der `apply-templates` Call selbst.

Beispielsweise möchte man die Titel im Kapitel anders behandeln als die Kapitel im Inhaltsverzeichnis, denn im TOC sollen z.B. keine Fussnoten-Marker angezeigt werden.

In Templates formuliert würde so eine Anweisung folgendermassen aussehen:

```
<xsl:template match="title" mode="toc">
  <div class="toc-entry">
    <xsl:apply-templates select="node()[not(self::footnote)]"/>
  </div>
</xsl:template>

<xsl:template match="title">
  <h1>
    <xsl:apply-templates/>
  </h1>
```

```
</xsl:template>
```

Die Generierung des TOC könnte dann so ablaufen:

```
<xsl:for-each select="chapter">
  <xsl:apply-templates select="title" mode="toc">
</xsl:for-each>
```

Bzgl. der Eindeutigkeit der Regelbasis kann man so anhand des Mode-Attributes Ausführungsgruppen bilden.

Wie auch bei Angabe der Priorities kann man auf diese Weise Regeln setzen, die nie ausgeführt werden, weil sie vllt. im Zuge einer Refactoring-Massnahme abgeklemmt und dann vergessen werden.

▻ Auch das *mode* -Attribut ist also mit Vorsicht zu geniessen und sparsam einzusetzen.

2.3 Namespaces

Wenn man XML Instanzen aus unterschiedlichen Quellen mit XSLT verarbeiten will, wird man sich wohl oder übel mit dem Thema Namespaces auseinander setzen müssen, um Konflikte in den Elementselektoren zu vermeiden.

Gerade bei hintereinandergeschalteten Transformationen kann es auch passieren, dass unerwartet ein Namespace in die Ausgabe generiert wird, den der folgende Prozesschritt nicht versteht, weil er dort nicht deklariert wurde.

Es gibt meherere Möglichkeiten einen Namespace im Stylesheet zu deklarieren. Gehen wir davon aus, dass in einem Transformationsschritt genau eine Quelle und max. eine Konfigurationsdatei verarbeitet wird, dann kann das Stylsheet-Element bspw. so aussehen:

```
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  xmlns:tektur="https://namespace-eigener-xslt-funktionen"
  xmlns="http://namespace-in-der-xml-eingabe.com/"
  xpath-default-namespace="https://namespace-der-konfigdatei.com/"
  exclude-result-prefixes="#all">
```

- Der *xsl* Namespace ist natürlich der Namespace für die XSLT Answeisungen und muss deklariert werden.

- Der `xs` Namespace ist notwendig, wenn man typisiert arbeiten will. Er erlaubt das Einbinden von Datentypen nach der XML Schema Spezifikation[36] und somit die bessere Validierung des Stylesheets zur Compile-Zeit.
- Die Deklaration eines eigenen geprefixten Namespaces erlaubt das Einbinden von eigenen XSLT Funktionen, wie z.B. auch das Einbinden der FunctX Bibliothek[37]
- Der Nicht-geprefixte Namespace ist der **Default-Namespace** und kann einen Namespace aus der Eingabe behandeln
- Das Attribut **xpath-default-namespace** gibt einen weiteren Namespace an, der in XPath Funktionen verwendet werden kann. In diesem Feld würde ich den Namespace einer Konfigurations- oder separaten Datendatei angeben.

Mehr als einen Namespace in der Eingabe sollte man aus meiner Sicht bei der XML Verarbeitung mit XSLT vermeiden - wenn es geht. Ggf. empfiehlt es sich, die Eingabe vor der Verarbeitung zu normalisieren und Elemente ggf. umzubenennen. Ansonsten kann man auch eigene Namespace-Prefixes deklarieren, wie z.B.:

```
xmlns:ext="https://www.tekturcms.de/external-tools"
```

und diesen in XPath Selektionen und Match-Regeln verwenden.

Befinden sich in den Eingabedaten Namespaces, die man in den XSLT Stylesheets nicht handelt - der Namespace kann auch nur an einem ganz bestimmten Element hängen - so kann es bei der Transformation - ohne Fehlermeldung - zu unerwarteten Ergebnissen kommen.

▻ Deshalb sollte man die Daten im Vorfeld bzgl. Namespaces sehr genau analysieren.

Namespaces in der Eingabe werden also meistens über die Kopfdeklaration in der Stylesheetdatei gehandelt. Welcher Namespace schliesslich in die Ausgabe geschrieben wird, hängt vom aktuell verarbeiteten Kontextknoten ab:

- Elemente, die man erzeugt, erhalten automatisch den Default-Namespace (wenn man nicht explizit einen Namespace angibt).
- Elemente, die man kopiert, transportieren den Namespace, den sie in der Eingabe hatten (wenn man dies nicht explizit verhindert).

Um diese beiden Default Einstellungen zu steuern (bzw. zu überschreiben) gibt es mehrere Möglichkeiten:

36) https://de.wikipedia.org/wiki/XML_Schema
37) http://www.xsltfunctions.com/

```
<xsl:element name="{local-name()}" namespace="{namespace-uri()}">
```

Hier wird ein Element mit dem un-geprefixten Namespace des Kontextknotens deklariert. Wenn der Kontextknoten keinen anderen Namespace hat, so wird hierdurch sichergestellt, dass der Default-Namespace auch tatsächlich in die Ausgabe kommt.

```
<xsl:element name="meinelement" namespace="mein-namespace">
```

Hier wird ein Element mit eigener Namespace Angabe in die Ausgabe geschrieben. Einfacher geschrieben:

```
<mein-element xmlns="mein-namespace">
```

Es gibt auch ein Attribut am *xsl:copy* Element, das den Vorgang des Namespace-Kopierens steuern kann:

```
<xsl:template match="p">
  <xsl:copy copy-namespaces="no">
     <xsl:apply-templates/>
   </xsl:copy>
</xsl:template>
```

Hier wird der Namespace am *p* Element nicht in die Ausgabe geschrieben.

Ebenso kann eine Default-Kopierregel verwendet werden, die es verbietet einen Namespace weiterzuvererben:

```
<xsl:template match="@* | node()">
   <xsl:copy inherit-namespaces="no">
     <xsl:apply-templates select="@* | node()"/>
   </xsl:copy>
</xsl:template>
```

Freie Wildbahn

In der freien Wildbahn bin ich erst kürzlich über folgendes Problem gestolpert:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xpath-default-namespace="https://tekturcms.de/schema/x12-xml/1.0"
                exclude-result-prefixes="#all"
                version="3.0">

  <xsl:template match="my-element">
    <neuer-name>
      <xsl:apply-templates>
    </neuer-name>
  </xsl:template>

  [...]
```

So deklariert würde das neue Element ***my-element*** mit einem leeren Namespace in die Ausgabe gesetzt, so ***<neuer-name xmlns=""/>*** . Das kann ein nachfolgender Transformationsschritt nicht lesen.

▻ Aus diesem Grund setze ich neue Elemente immer mit dem Element-Konstruktor in die Ausgabe.

Beispiel:

```
<xsl:template match="my-element">
    <xsl:element name="neuer-name" namespace="{namespace-uri()}">
      <xsl:apply-templates>
    </xsl:element>
  </xsl:template>
```

Namespaces in XQuery

Während XSLT dazu dienen sollte, XML Daten in andere (XML-) Formate zu transformieren, dient XQuery z.B. dazu auf einer **NoSQL** Datenbank Daten aus unterschiedlichen Quellen zu selektieren, zu harmonisieren und an verarbeitende Prozesse weiterzugeben.

Deshalb ist es für mich nicht so erstaunlich, dass das Namespace Konzept in XQuery irgendwie besser funktioniert.

Damit man überhaupt Daten auf einem mit Namespaces versehenen XML Dokument selektieren kann, müssen alle Namspaces am Anfang des XQuery Ausdrucks angegeben werden, das sieht so aus:

```
xquery version "1.0-ml";

import module namespace tektur = "http://www.teturcms.de/xquery/common"
                                 at "common.xqy";
import module namespace mem = "http://xqdev.com/in-mem-update"
                                 at '/MarkLogic/appservices/utils/in-mem-update.xqy';
declare namespace local = "https://tekturcms.de/code/alex-sandbox/1.0";
declare namespace weiredns = "https://weired-ns-in-input-data.com/weired/ns";
declare namespace xs = "http://www.w3.org/2001/XMLSchema";
```

Hier werden zuerst Funktionen aus anderen Modulen eingebunden. Diejenigen in einer eigenen Datei `common.xqy` , sowie aus der Bibliothek `mem` in der **MarkLogic** Umgebung. Danach wird ein Namespace `local` für eigene Funktionen innerhalb der Quelldatei deklariert, sowie der Namespace `weiredns` , der in den Eingabedaten vorhanden ist. Der Namespace `xs` ist analog zum XSLT Beispiel gesetzt.

2.4 Schemata

XML Daten können sehr komplex werden. Da ihre Eingabe oft durch keine User-Interface Massnahmen oder sonstigte Regelungen beschränkt ist, sie im Prinzip mit

jedem Texteditor verändert werden können und gewöhnlich über viele Stationen verschickt / verarbeitet werden, ist es ratsam deren formale und inhaltliche Korrektheit zu überprüfen.

Dazu wird die gute maschinelle Lesbarkeit der XML Daten von Validierungsengines ausgenützt. Es gibt viele unterschiedliche Schematypen, gegen die validiert werden kann. Das sog. *Schema* ist dann ein Satz von Regeln, der beim Baumdurchlauf abgeglichen wird.

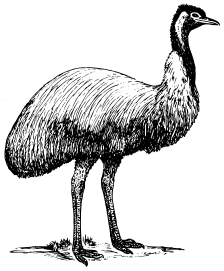
Angefangen hat wohl alles mit der Dokumenttypdefinition (DTD[38]) die sowohl für XML Daten als auch für den Vorläufer SGML angewendet werden kann.

Die mangelnde Fähigkeit zur Überprüfung von sematischen Beziehungen mittels DTD, hat wohl zu XML Schema[39] geführt.

XML Schema ist konsequent mittels XML modelliert und hat dieselbe gute Eigenschaft bzgl. der Maschinenlesbarkeit, wie XML an sich. Somit können bei einem Baumdurchlauf auch komplexe Regeln beim Abgleich von Datenstruktur und Valdierungsregel maschinell überprüft werden.

Vllt. hat sich aber gerade dieser Vorteil, nämlich die Maschinenlesbarkeit, im Laufe der Zeit als Nachteil herauskristallisiert. Es gibt zwar einige sehr gute visuelle Modellierungswerkzeuge, die es erlauben die Regeln als einen Baum grafisch zu modellieren, sobald aber komplexere Beziehungen modelliert werden sollen, ist man mit diesen Tools ein bisschen gefangen und man wünscht sich doch wieder die Flexibilität eines Texteditors.

Grafische Werkzeuge sind bspw.:



- Ein Tool, das aussieht wie aus einem anderen Jahrhundert: Der Near & Far Designer[40]
- TreeVision[41] von der Ovidius GmbH in Berlin.
- Visual Schema Editor[42] als Teil des oXgen XML Editors.

38) https://de.wikipedia.org/wiki/Dokumenttypdefinition
39) https://de.wikipedia.org/wiki/XML_Schema
40) http://www.perfectxml.com/SoftDetails.asp?id=216
41) https://www.gds.eu/de/redaktionssysteme/weitere-loesungen
42) https://www.oxygenxml.com/xml_editor/rng_visual_schema_editor.html

⊳ Schemata

Eine DTD wird im Near&Far Designer als aufklappbare Baumstruktur angezeigt. So könnnen auch sehr komplexe DTDs mit 1000 Elementen effizient untersucht werden. Über einen Eingabedialog lassen sich Attribute hinzufügen oder ändern. Auch das Neuanlegen von einzelnen Zweigen (= Hinzufügen von Elementen) lässt sich grafisch erledigen. Jedoch ist es ratsam, sich zumindest das Grundgerüst der DTD mit einem Texteditor zu überlegen.

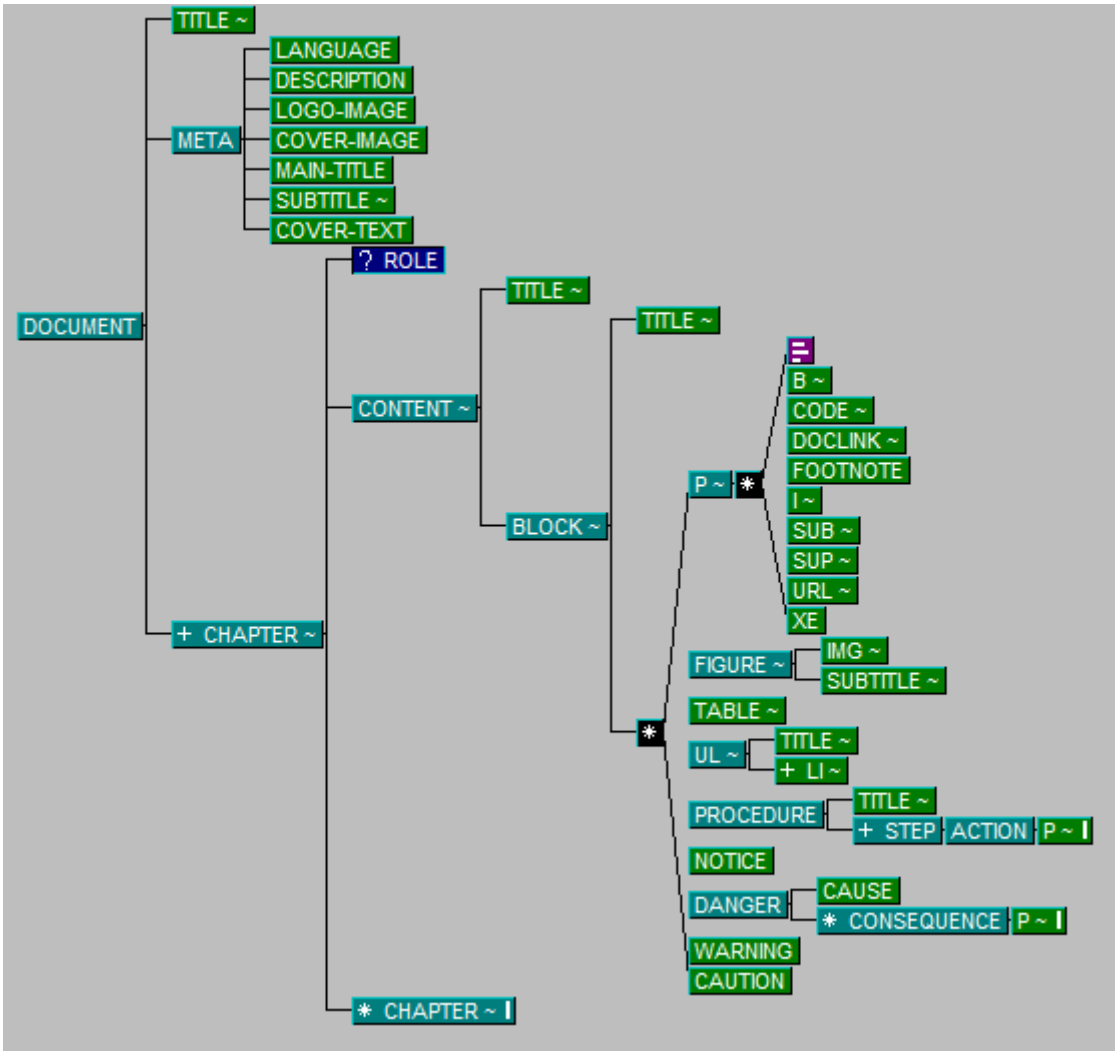


figure 4: Abbildung einer DTD Struktur im Near&Far Designer

Ähnlich wie in Programmierumgebungen zählt aber schlussendlich, wie schnell man etwas in umfangreichen Quelltexten wiederfindet und anpassen / erweitern kann. Hier ist nach-wie-vor Plain-Text unschlagbar.

Deshalb ist es nicht verwunderlich, dass sich (wieder) leichtgewichtige Validierunsformate etablieren, die sich schön mit einem Texteditor editieren lassen, wie z.B. RelaxNG. RelaxNG existiert zwar schon seit 2002, erfreut sich aber in letzter Zeit zunehmender Beliebtheit.

Die Kompaktform der Regeln von RelaxNG sieht ein bisschen aus, wie JSON - was gerade für Webentwickler interessant sein könnte. Zudem wird die zugrunde liegende Logik der Backus-Naur Form[43)] relativ klar herausgestellt, was die Regelfindung erleichtert.

43) https://de.wikipedia.org/wiki/Backus-Naur-Form

Um einen ersten Eindruck von der Syntax zu bekommen, habe ich mal das erste Beispiel aus dem RelaxNG Tutorial[44] gestolen:

Consider a simple XML representation of an email address book:

```
<addressBook>
    <card>
        <name>John Smith</name>
        <email>js@example.com</email>
    </card>
    <card>
        <name>Fred Bloggs</name>
        <email>fb@example.net</email>
    </card>
</addressBook>
```

The DTD would be as follows:

```
<!DOCTYPE addressBook [
<!ELEMENT addressBook (card*)>
<!ELEMENT card (name, email)>
<!ELEMENT name (#PCDATA)>
<!ELEMENT email (#PCDATA)>
]>
```

A RELAX NG pattern for this could be written as follows:

```
<element name="addressBook" xmlns="http://relaxng.org/ns/structure/1.0">
    <zeroOrMore>
        <element name="card">
            <element name="name">
                <text/>
            </element>
            <element name="email">
                <text/>
            </element>
        </element>
    </zeroOrMore>
</element>
```

In der Kurzform würde dieses Beispiel so aussehen:

```
element addressBook {
    element card {
        element name { text },
        element email { text }
    }*
}
```

Es gibt auch ein Tutorial zur Kurzform[45].

44) https://relaxng.org/tutorial-20011203.html#IDAHDYR
45) https://relaxng.org/compact-tutorial-20030326.html

▻ Schemadateien, die in der Kurzform verfasst sind, tragen gewöhnlicherweise die Dateiendung `.rnc` . Dateien in der Langform haben die Endung `.rng` .

Mit RelaxNG kann man Schema-Grammatiken fast so elegant wie mit der Backus Naur Normalform - BNF[46]modellieren - wie man das im Informatik Unterricht gelernt hat.

Exklusion mit RNC

Das schöne an RelaxNG ist die Tatsache, dass man damit Sachen machen kann, die mit anderen Schemasprachen nicht so leicht gehen. Z.b. kann man ein unvollständiges Schema erzeugen, das nur ganz bestimmte Teile der XML Instanz prüft.

Betrachten wir dazu das folgendes Beispiel:

```
unbehandeltesElement =
    element * - (aussteller |
                 empfaenger |
                 datum) {
        (attribute * { text } |
         text |
         unbehandelteElemente)*
    }

start =
    element abrechung {
        element id { xsd:NMTOKEN },
        element datum  { xsd:date },
        [...]
        element zahlungen {
            element zahlung {
                element id { xsd:NMTOKEN },
                element datum  { xsd:date },
                element plan { xsd:NMTOKEN },
                [...]
            },
            element beleg-daten {
                element beleg {
                    attribute nummer { xsd:integer }?,
                    element datum  { xsd:date },
                    unbehandeltesElement+,
                    element aussteller { text },
                    unbehandeltesElement+,
                    element empfaenger { text },
                    unbehandeltesElement+
                }
            }
        }+
    }
}
```

Von den Elementen *`abrechnung`* und *`zahlung`* wissen wir, wie sie aufgebaut sind und können sie vollständig modellieren. Die unbekannte Größe ist allerdings das Element *`beleg`* . Dieses stammt von einer externen Quelle, und wir wissen nur, das darin zwingend die Felder *`datum`* , *`empfaenger`* und *`aussteller`* vorhanden sein müssen.

Zwischen diesen Elementen gibt es mindestens ein, aber auch mehrere unbekannte Elemente. Damit wir nun XML Instanzen, die nach diesem Schema aufgebaut sind, validieren können, wird ein Element *`unbehandeltesElement`* modelliert, das sozusagen einen Platzhalter darstellt. Dieses schliesst explizit die zwingenden Felder

46) https://de.wikipedia.org/wiki/Backus-Naur-Form

datum , *empfaenger* und *aussteller* aus, um deren Validierung durch das Schema nicht zu verfälschen.

Relaxtron

Besonders fortschrittlich klingt die Möglichkeit in ein RelaxNG Schema weitere Schematron-Regeln einzubinden - Relaxtron[47]. Damit kann man bspw. sicherstellen, dass bestimmte Constraints bzgl. der Elementstruktur eingehalten werden. Mit einer Schemasprache allein ginge das nicht. Betrachten wir das folgende Stück RelaxNG in der Kompaktform:

```
namespace sch = "http://purl.oclc.org/dsdl/schematron"

start = ul
ul =
  [
    sch:pattern [
      sch:rule [
      context = "list"
        sch:assert [
          test = "not(ancestor::ul and ancestor::procedure)"
                 'Eine ul darfin der Procedure nicht verschachtelt werden!'
        ]
      ]
    ]
  ]
  element ul {
    any_attribute*,
    listitem+
  }

listitem =
  element listitem {
    any_attribute*,
    ( para | ul )+
  }

para =
  element para {
    any_attribute*,
    text*
  }
any_attribute = attribute * { text }
```

Hier wird sichergestellt, dass eine ungeordnete Liste *list* nicht tiefer verschachtelt wird - aber nur innerhalb einer *procedure* .

Der oXygen Editor kann die RelaxNG Kopaktform in die XML Form überführen und schön visualisieren. Das sieht dann so aus:

47) https://www.xml.com/pub/a/2004/02/11/relaxtron.html

Schemata

Im oXygen Editor gibt es eine Vollmodell-Ansicht und eine Logische Modellansicht für RelaxNG Schemas, die auch eingebettete Schematron-Knoten anzeigen. Mit Klick auf ein Symbol gelangt man zum betreffenden Quelltextstück.

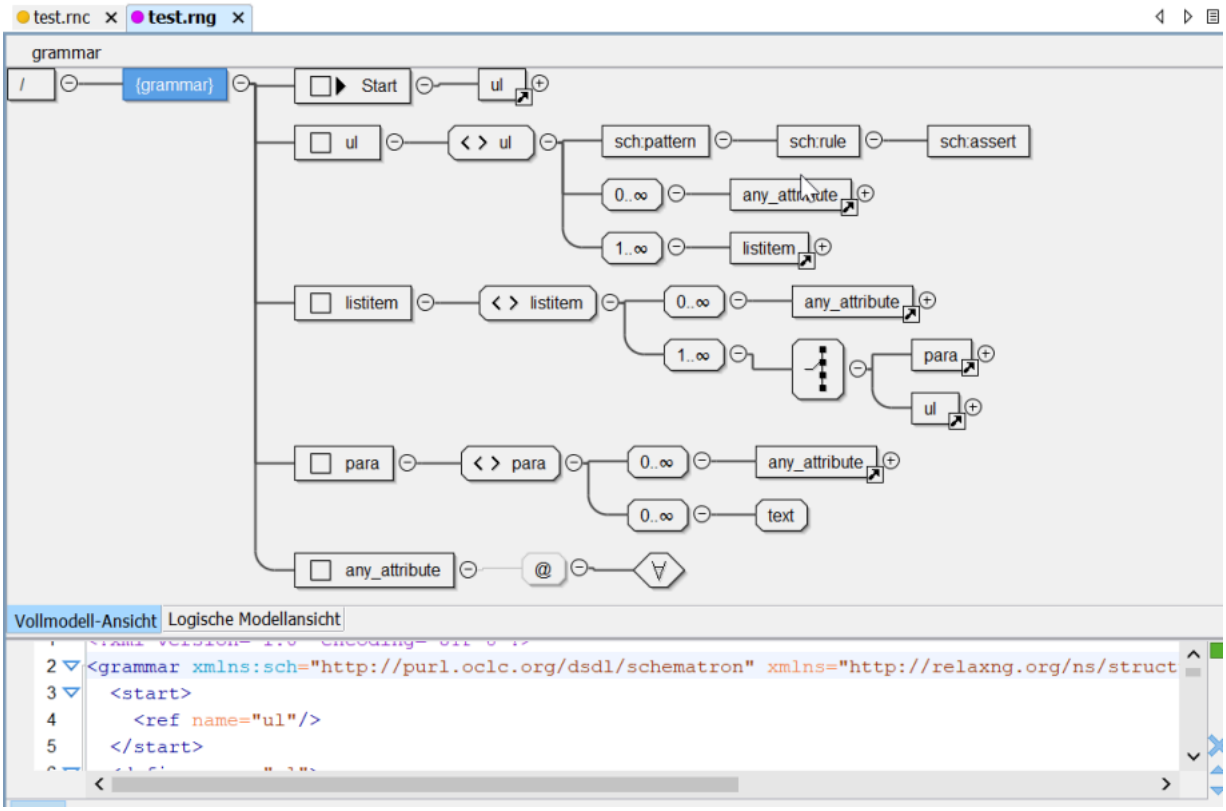


figure 5: Anzeige eines RelaxNG Schemas im oXygen Editor

Angabe des Schemas in der XML Instanz

Wie auch schon im Kapitel Anwendungsgebiete on page 7 beschrieben, werden alle zusätzlichen - für den XML Prozessor wichtigen - Instruktionen, die nicht Teil der Daten sind, über eine *Processing Instruction* in das XML eingebunden.

Es können auch mehrere Schemata mittels einer Processing Instruction eingebunden werden, so dass eine Validierung über mehrere Engines möglich ist.

Übernimmt man bspw. einen Legacy Datenbestand, für den nur eine DTD vorhanden ist, so kann im Nachhinein auch noch ein XML Schema hinzugefügt werden, welches komplexere semantische Beziehungen überprüft.

Für eine weitere Überprüfung steht schliesslich Schematron, vgl. Kapitel Validierung mit Schematron on page 148 zur Verfügung. Ein Beispiel, wie man mehrere Schemas angibt, findet sich auf der Seite des W3 Konsortiums zum Thema[48)]

```
<?xml version="1.0"?>
<?xml-model href="http://www.docbook.org/xml/5.0/rng/docbook.rng"?>
<?xml-model href="http://www.docbook.org/xml/5.0/xsd/docbook.xsd"?>
<book xmlns="http://docbook.org/ns/docbook">
  [...]
</book>
```

48) https://www.w3.org/TR/xml-model/

Zum Einbinden einer RelaxNG Kompaktform genügt es bspw. diese PI (Processing Instruction) anzugeben

```
<?xml-model href="whatever.rnc" type="application/relax-ng-compact-syntax"?>
```

oXygen und auch alle anderen Editoren unterstützen die Angabe des Schemas mittels *PI*, so dass man auch im Editor ohne explizite Anfrage validieren kann.

oXygen Validierung

Die Validierungswerkzeuge finden sich im oXygen Editor unter `Document` > `Validate` > `Validate With`

Über das `Document` Tab findet man in oXygen die Validierungsoptionen.

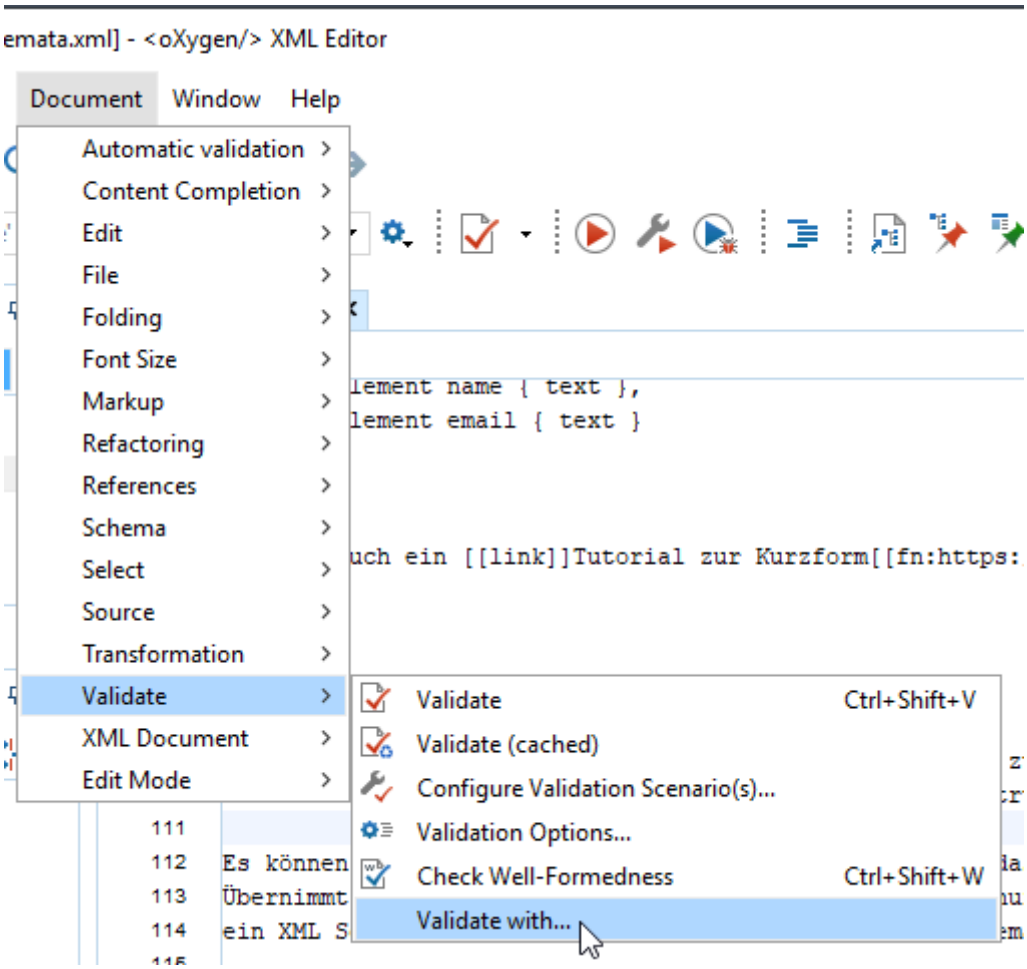


figure 6: Validierungsdialog in oXygen

Es lassen sich mit oXygen auch Schemas konvertieren, `Document` > `Schema` > `Generate/Convert Schema`

Tools zur Konvertierung sind ebenfalls im *Document* Tab untergebracht

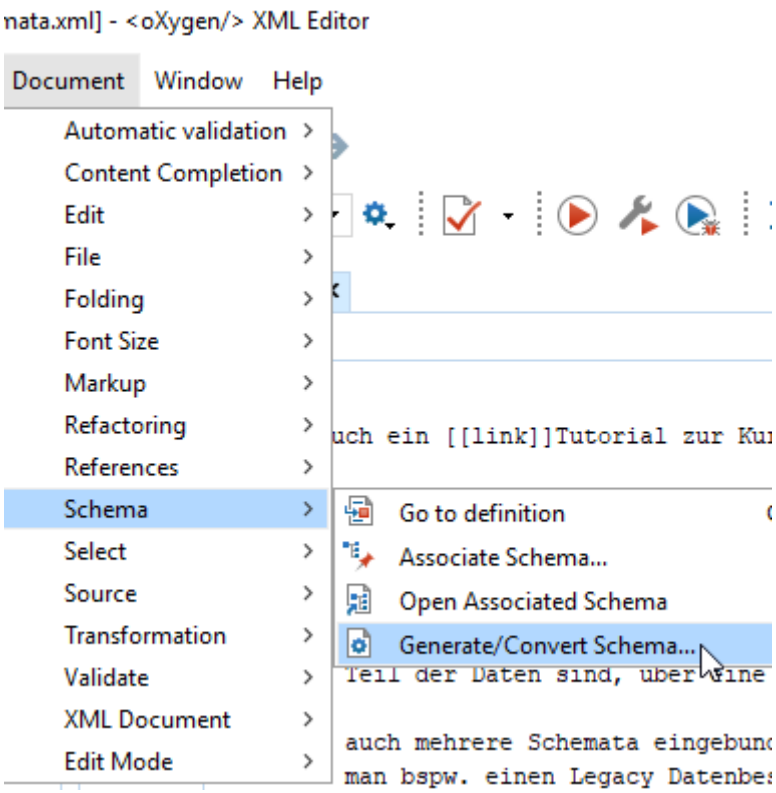


figure 7: Schema-Konvertierung mit oXygen

Jing und Trang

Für RelaxNG gibt es den Jing Validator[49]. Bemerkenswert ist hier, dass das Tool von James Clark[50] entwickelt wird (ein Urgestein in der XML Datenwelt).

Passt zwar nicht ganz zum Thema: Ein Tool von James Clark, mit dem ich schon öfters gearbeitet habe, ist SP[51] ein SGML System, mit dem man SGML nach XML konvertieren kann. Diese Funktionalität ist wohl sonst eher selten zu finden ...

Um eine XML Instanz mit Jing zu validieren genügt folgender Kommandozeilenaufruf:

49) https://relaxng.org/jclark/jing.html
50) http://www.jclark.com/bio.htm
51) http://www.jclark.com/sp/

```
jing resources/schemas/person.rng resources/examples/person.xml
```

Jing unterstützt nicht die RelaxNG Kompaktform!

- Die Kompaktform muss zuerst in die Normalform mit Trang[52] konvertiert werden.

Um die Kompaktform von RelaxNG in die Normalform zu konvertieren setzt man diesen Befehl auf der Kommandozeile ab:

```
trang -I rnc -O rng resources/schemas/person.rnc resources/schemas/person.rng
```

Schema Single-Sourcing

Das Single-Sourcing Konzept macht auch vor Schemata nicht halt. Aus einer Quelle in der RelaxNG Kompaktform können alternative Ansichten in anderen Schema-Sprachen erzeugt werden, wie bspw. XML Schema (XSD)[53].

Mit einem BPMN Diagramm kann man das Single-Sourcing Konzept bei der Schema-Erzeugung recht gut veranschaulichen. Aus einer *RNC* Quelle wird die RelaxNG XML Form, die Schematron-Regeln, die XSD und das XSLT der Schematron-Regeln generiert. Diese Artefakte dienen zur Validierung im Editor und zur Validierung in einer Transformer-Pipe.

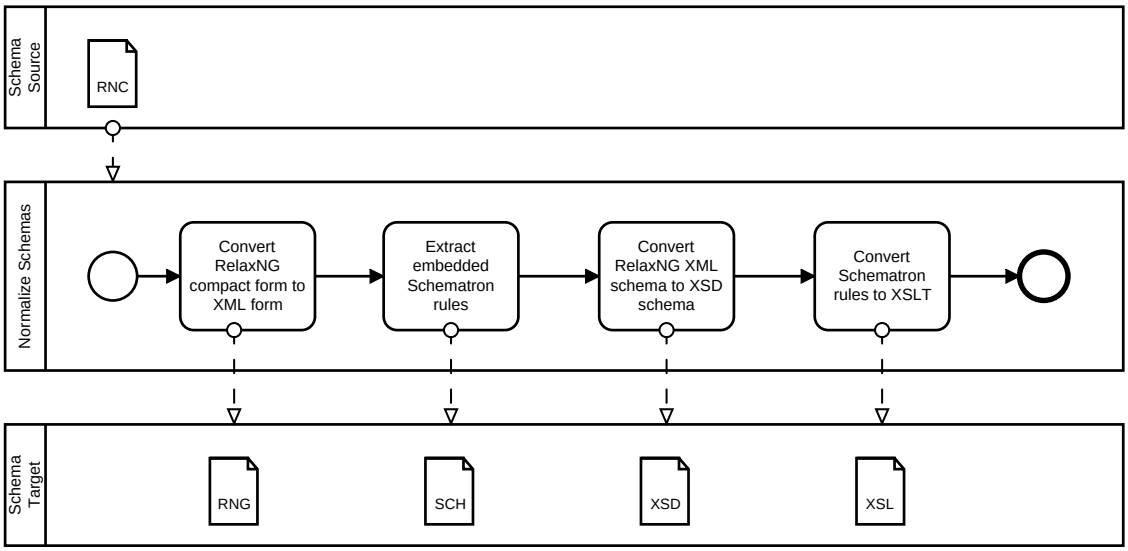


figure 8: BPMN Diagramm zum Single-Sourcing Konzept bei RelaxNG Schemas

52) https://relaxng.org/jclark/trang.html
53) https://de.wikipedia.org/wiki/XML_Schema

Ein Shell-Skript, das den Prozess aufruft, könnte z.B. so aussehen:

```
#!/bin/sh

echo "converting from RelaxNG compact form to RelaxNG XML form..."
trang -I rnc -O rng test.rnc test.rng

echo "extracting embedded schematron rules from RelaxNG schema..."
saxon -s:test.rng -o:test.sch -xsl:rng2sch.xsl

echo "converting RelaxNG schema to XML Schema..."
trang -O xsd test.rng test.xsd

echo "converting schematron rules to XSLT..."
saxon test.sch validation/schematron/iso_dsdl_include.xsl | \
saxon -s:- validation/schematron/iso_abstract_expand.xsl | \
saxon -s:- validation/schematron/iso_svrl_for_xslt2.xsl > test.xsl

echo "Done!"
```

Ein unpoliertes Skript zum Extrahieren der Schematron Regeln aus der RNC habe ich auf Github gefunden[54].

2.5 Standards

Schemata wären ohne Standards nur ein Mittel um die syntaktische und strukturelle Korrektheit der XML Daten zu überprüfen - in einem unternehmensweiten Kontext.

Sollen die Daten dagegen über Unternehmensgrenzen hinweg kommuniziert werden, so empfielt sich die Informationsarchitektur bzw. das Schemata an einem Standard auszurichten, der auch anderen gut zugänglich ist.

Wie die Praxis zeigt, gibt es aber immer irgendwelche anwendungsspezifischen Sondertüten, die nicht von einem Standard abgefangen werden können.

Deshalb ist darauf zu achten, dass der Standard anwendungsspezifisch erweitert werden kann, ohne dabei die standardisierte Information zu verfälschen.

2.5.1 DITA

DITA[55] ist so ein Standard für die Technische Dokumentation. Allgemeine Teile können von allen DITA Systemen verarbeitet werden - also alle Teile der Information, die für die Aussenwelt verfügbar gemacht werden sollen, das ist z.B. eine Zulieferdokumentation für bestimmte Maschinenbauteile.

Spezifische Information, wie z.B. Referenzen in eine interne Datenbank, würde dagegen ein externer DITA Prozessor entweder verschlucken oder anderweitig kennzeichnen - je nachdem wie die Abarbeitung von unbekannten Elementen im externen System realisiert ist.

DITA vs Docbook

Grob betrachtet ähnelt die DITA Struktur derer von Docbook[56]. Sie hat die typische Kapitelverschachtelung.

Zum Vergleich ist im folgenden ein Docbook Grundgerüst und ein DITA Grundgerüst (Wikipedia) abgebildet.

54) https://github.com/citation-style-language/utilities/blob/master/RNG2Schtrn.xsl
55) https://de.wikipedia.org/wiki/Darwin_Information_Typing_Architecture
56) https://de.wikipedia.org/wiki/DocBook

Docbook Grundgerüst

```
<book id="einfaches_buch"
    <title>Ein sehr einfaches Buch</title>
    <chapter id="einfaches_kapitel">
        <title>Kapitel</title>
        <para>Hallo Welt!</para>
    </chapter>
</book>
```

DITA Grundgerüst

```
<topic id="maintaining" xml:lang="en-us">
   <title>Maintaining</title>
   <shortdesc>
      You maintain your solution to ensure that all components
      are operating at maximum efficiency.
   </shortdesc>
   <body>
      <p>
         Maintenance is a task that you perform along
  with configuration to get the most from your solution.
      </p>
   </body>
</topic>
```

Während beim Docbook Standard noch ein Element *<book>* die einzelnen Kapitel klammert, ist die Vorgehensweise beim DITA Standard topic-basiert - Stichwort: Topic Based Authoring.

Hier konzentriert sich der Redakteur nicht mehr so sehr auf das Buch als Ganzes, sondern mehr auf die Abgeschlossenheit und Referenzierbarkeit der einzelnen Topics (= Informationseinheiten), um diese in einer anderen Publikation ggf. in einem anderen Format (Online, Print, Smartphone) leicht wiederverwenden zu können.

Im obigen XML Schnippsel vermutet man z.B., dass das Element *<shortdesc>* in einer elektronischen Publikation auf einer Übersichtsseite angezeigt werden könnte (- um die Referenz zum eigentlichen Topic zu charakterisieren), während in einer Print-Publikation diese Information als einleitender Kurztext gedruckt werden könnte.

Was ist aber nun der Clou beim DITA Standard? Dokumenttyp Definitionen (DTDs), die einige Elemente umbenennen und ein paar semantische Besonderheiten bereitstellen, gibt es viele.

Der Clou bei DITA

Ich denke, das Besondere bei DITA sind ein paar technische Kniffe, die eine gewisse Generizität des Ansatzes realisieren. Wenn wir im obigen XML Schnippsel noch **die DTD mit angeben**:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE topic SYSTEM "ditadtd/topic.dtd">
<topic id="maintaining" xml:lang="en-us">
   [...]
</topic>
```

und eine Identity-Transformation ausführen, so sind magischerweise weitere Attribute hinzugekommen, vgl.

```
<?xml version="1.0" encoding="UTF-8"?>
```

```
<topic xmlns:ditaarch="http://dita.oasis-open.org/architecture/2005/"
       id="maintaining" xml:lang="en-us"
       ditaarch:DITAArchVersion="1.3"
       domains="(topic ui-d) (topic hi-d) (topic pr-d) (topic sw-d)
               (topic ut-d) (topic indexing-d)"
       class="- topic/topic ">
    <title class="- topic/title ">Maintaining</title>
    <shortdesc class="- topic/shortdesc ">
        You maintain your solution to ensure that all components
        are operating at maximum efficiency.
    </shortdesc>
    <body class="- topic/body ">
        <p class="- topic/p ">
            Maintenance is a task that you perform along
            with configuration to get the most from your solution.
        </p>
    </body>
</topic>
```

Besonders interessant ist hier das `@class` Attribut. Wie ist das nun passiert?

> ▶ DITA definiert eine Reihe von `#FIXED` Attributen[57] mit einem festen Wert, die der Parser beim Validieren automatisch an die Elemente hängen muss. Ungeparst sieht ein Topic recht einfach aus. Geparst kann das DITA XML dagegen recht aufgebläht wirken.

XSLT Regeln in einem DITA Prozessor matchen nun nicht auf die Elementnamen, sondern auf die `@class` Attribute, z.B. so:

```
<xsl:template match="*[contains(@class,' topic/topic ')]/
                     *[contains(@class,' topic/title ')]">
[...]
```

Hier soll der Titel des Topics formatiert werden. In einer Spezialisierung[58] des Topics - das wäre z.B. ein Concept:

```
<?xml version="1.0" encoding="UTF-8"?>
<!DOCTYPE concept PUBLIC "-//OASIS//DTD DITA Concept//EN" "concept.dtd">
<concept id="concept_kl3_knd_f3b">
    <title>My concept</title>
    <shortdesc>Just a concept to illustrate specialization</shortdesc>
    <conbody>
        <p>Yeah, we need some content here, too!</p>
    </conbody>
</concept>
```

werden nun die `#FIXED` Attribute folgendermassen gesetzt:

```
<?xml version="1.0" encoding="UTF-8"?>
<concept xmlns:ditaarch="http://dita.oasis-open.org/architecture/2005/"
         id="concept_kl3_knd_f3b"
         ditaarch:DITAArchVersion="1.3"
         class="- topic/topic concept/concept ">
    <title class="- topic/title ">My concept</title>
    <shortdesc class="- topic/shortdesc ">Just a concept to
```

57) https://wiki.selfhtml.org/wiki/XML/DTD/Attribute_und_Wertzuweisungen
58) https://www.ibm.com/developerworks/library/x-dita2/index.html

```
        illustrate specialization</shortdesc>
    <conbody class="- topic/body  concept/conbody ">
        <p class="- topic/p ">Yeah, we need some content here, too!</p>
    </conbody>
</concept>
```

Unsere Regel von oben mit

```
<xsl:template match="*[contains(@class,' topic/topic ')]/
                     *[contains(@class,' topic/title ')]">
[...]
```

die einen Titel formatiert, würde also sowohl für Topic-Elemente, als auch für Concept-Elemente gelten. Will man eine Spezialformatierung für Concept-Elemente, so müsste man die folgende Regel noch zur Regelbasis mithinzunehmen:

```
<xsl:template match="*[contains(@class,' concept/concept ')]/
                     *[contains(@class,' topic/title ')]">
[...]
```

Warum ist das Ganze nun so clever? Wenn bspw. Firma XYZ noch keine Concept-Definitionen erstellt hat, kann sie trotzdem Concept-Elemente der Firma ABC verarbeiten, weil die Match-Regeln nicht auf Elementnamen operieren, sondern auf dynamisch durch die DTD hinzugefügte Klassenattribute - die neben der Bezeichner der Spezialisierung auch die Bezeichner der Generalisierung enthalten. Diese kennt jeder DITA Prozessor und kann zumindest einen Default bereitstellen.

Für obiges Beispiel - bzgl. des Titels - würde das bedeuten, dass der Concept-Titel in Firma XYZ nicht so schön (oder an einer bestimmten Stelle) dargestellt wird, wie er in Firma ABC angedacht war. Der wichtige Content wird aber trotzdem dargestellt.

▶ Die Referenzimplementierung für DITA ist übrigens das DITA Open Toolkit[59)]

Das Erstellen einer redundanzarmen DITA DTD ist allerdings eine Wissenschaft für sich. Schliesslich will man ja auch für eigene Spezialisierungen den generischen Ansatz beibehalten. Der DITA Redakteur oder besser der Dokumentationsbeauftragte muss sich deshalb einer ziemlich komplizierten Namens- und Strukturkonvention unterziehen.

2.5.2 DITA Inhaltsmodell

An dieser Stelle sei kurz auf das umfangreiche Inhaltsmodell von DITA in Form von XML Schnippseln eingegangen. Gerade soviel DITA wie nötig ist, um mit Tektur CCMS arbeiten zu können.

59) https://github.com/dita-ot/dita-ot

Inlineelemente

Buchwelt	DITA
Paragraph	Paras erlauben die gängigen Inlineelemente, z.B. <b> , <i> , <u> , <sup> , <sub> , und werden mittels <p> Tag ausgezeichnet. `<p>Das ist ein Para mit <b>fettem</b> und <i>kursiven</i> Text, sowie mit hochgestellter<sup>Quadratzahl</sup> und tiefgestellter<sub>Zahlenuntergrenze</sub>. </p>`
Fußnote	Fußnoten werden mittels <fn> Tag ausgezeichnet. Das Attribut @callout ist optional und gibt den Fußnotenschlüssel an. Ohne Angabe wird die Fußnote meist fortlaufend nummeriert. `<fn callout="#1"> Das ist Textin einer Fußnote. Hier sind auch Links erlaubt <link href="http://www.tekturcms.de">Tektur Homepage</link> </fn>`
Link	Links können sowohl extern auf eine Ressource im Internet verweisen, aber auch intern auf einen anderen Topic. Hierbei würde die Syntax so aussehen: `<link href="#topicid" format="dita">Link auf anderen Topic</link>`
Indexeintrag	Indexeinträge werden mit dem <indexterm> Element ausgezeichnet. Sie können u.a. direkt im Content vorkommen und bei Tektur CCMS auch einen verschachtelten Untereintrag aufweisen. `<indexterm>Verarbeitungsmethoden <indexterm>Vortransformation</indexterm></indexterm>`
Icon	Inzeilige Icons können über das Element <image> bedatet werden. `<p>Jetzt kommt ein Icon <image href="icon.png"/>.</p>`
Code	Inzeilige Codeschnippsel oder Dateipfade, etc. werden über das Element <codeph> gesetzt. `<p>Windows Fonts Ordner: <codeph>C:\Windows\Fonts</codeph>.`

Blockelemente

Buchwelt	DITA

Grafik	Die Grafik kann seitenbreit, spaltenbreit oder in der Marginalie angebracht werden. Das wird in Tektur CCMS über das Attribut @expanse gesteuert.
	
	`<fig expanse="column" frame="all" scale="50">`
	`  <image href="Cormorant.svg"/>`
	`</fig>`

Quelltext	Quelltexte oder vorformatierter Text können einerseits inzeilig über das *<codeph>* Element bedatet werden oder als Blockelement mittels des *<pre>* Elements.
	
	`<pre xml:space="preserve">`
	`for iin range(10):`
	`  print i`
	`</pre>`

Warnhinweis	Warnhinweise werden mittels des Elements *<hazardstatement>* bedatet. Es gibt vier verschiedene Ausprägungen:
	
	– warning
	– danger
	– caution
	– notice
	
	Diese wird über das Attribut *@type* gesteuert.
	
	`<hazardstatement type="warning">`
	`  <messagepanel>`
	`  <typeofhazard>`
	`  Heisse Flächen sollte nicht berührt werden.`
	`  </typeofhazard>`
	`  <howtoavoid>Warnschilder anbringen</howtoavoid>`
	`  <howtoavoid>Temperaturregler einschalten</howtoavoid>`
	`  </messagepanel>`
	`</hazardstatement>`
	
	Der Beschreibung der Gefahrenstelle *<typeofhazard>* können dabei mehrere Maßnahmen (*<howtoavoid*) folgen.

Hinweis

Der einfache Hinweis wird über das Element *<note>* bedatet.

```
<note>
    <p>Ohne weitere Maßnahmen werden Dokumente
       mit den Rechten des Erzeugers versehen.</p>
</note>
```

Listen

Es stehen geordnete, ungeordnete und Definitionslisten zur Verfügung. *<ul>* und *<ol>* Listen können analog zu bspw. HTML gesetzt werden. Zusätzlich erlaubt DITA auch Paras in Listenpunkten. Die folgenden Beispiele sind von der DITA Dokupage[1]:

```
<ol>
    <li>Red</li>
    <li>Orange</li>
    <li><p>Yellow</p></li>
    <li>Green</li>
    <li>Blue</li>
    <li>Indigo</li>
    <li>Violet</li>
</ol>
<ul>
    <li>This is an itemin an unordered list.</li>
    <li>To separate it from other itemsin the list,
        the formatter puts a bullet beside it.</li>
     <li>The following paragraph, containedin the list item
         element, is part of the list item which contains it.
         <p>This is the contained paragraph.</p>
     </li>
     <li>This is the last list itemin our unordered list.</li>
</ul>
```

Außerdem steht noch die Definitionsliste *<dl>* zur Verfügung mit der auch dieses Kapitel realisiert ist:

```
<dl>
    <dlhead>
        <dthd>Image File View Selection</dthd>
        <ddhd>Resulting Information</ddhd>
    </dlhead>
    <dlentry>
        <dt>File Type</dt>
        <dd>Image's file extension</dd>
    </dlentry>
    <dlentry>
        <dt>Image Class</dt>
        <dd>Image is raster, vector, metafile or 3D</dd>
    </dlentry>
    <dlentry>
        <dt>Number of pages</dt>
        <dd>Number of pagesin the image</dd>
    </dlentry>
    <dlentry>
        <dt>Fonts</dt>
        <dd>Fonts contained within a vector image</dd>
    </dlentry>
</dl>
```

1) https://docs.oasis-open.org/dita/v1.0/langspec/ul.html

Strukturelemente

Buchwelt	DITA

Kapitel	Ein Topic sollte eine in sich abgeschlossene Informationseinheit bilden, die in verschiedenen Kontexten eingebunden werden kann.
	
	`<topic>`
	`  <title>Der Titel des Topics</title>`
	`  <body>`
	`  <p>Ein einleitender Satz vor der Grafik</p>`
	`  <fig>`
	`  <image href="schöne_grafik.svg"/>`
	`  </fig>`
	`  <p>Ein abschliessender Satz zum Topic</p>`
	`  </body>`
	`</topic>`

Abschnitt	Eine Section ist ein Abschnitt mit einer Unterüberschrift innerhalb eines Topics.
	
	`<section>`
	`  <title>Unterabschnitt</title>`
	`  <p>Blockelemente, wie Paras, folgen direkt dem Titel</p>`
	`</section>`

Kapitelstruktur	Die Kapitelstruktur wird in DITA mittels einer verschachtelten Topic-Struktur definiert, die sogenannte DITA Map. Hierbei werden *<topicref>* Elemente verwendet, die mittels des Attributs *@href* auf vorhandene Topics verweisen. Die Titel der einzelnen Topics sind im Attribut *@navtitle* zur Erzeugung des Inhaltsverzeichnisses vorgehalten.
	
	`<?xml version="1.0" encoding="utf-8"?>`
	`<map title="XML Entwicklerhandbuch">`
	`  <topicmeta>`
	`  <navtitle>XSLT - XQuery - MarkLogic</navtitle>`
	`  <shortdesc/>`
	`  <keywords>`
	`  <keyword>XML</keyword>`
	`  <keyword>XSLT</keyword>`
	`  <keyword>XQUERY</keyword>`
	`  </keywords>`
	`  </topicmeta>`
	`  <topicref href="1_Intro.dita" navtitle="Intro"/>`
	`  <topicref href="2_Anwendungsgebiete.dita"`
	`  navtitle="Anwendungsgebiete">`
	`  <topicref`
	`  href="2_1_XSLT___die_Programmiersprache_im_XML_Bereich.dita"`
	`  navtitle="XSLT - die Programmiersprache im XML Bereich"/>`
	`  <topicref`
	`  href="2_2_Aktuelle_und_vergangene_Anwendungen.dita"`
	`  navtitle="Aktuelle und vergangene Anwendungen"/>`
	`[...]`

Taskelemente

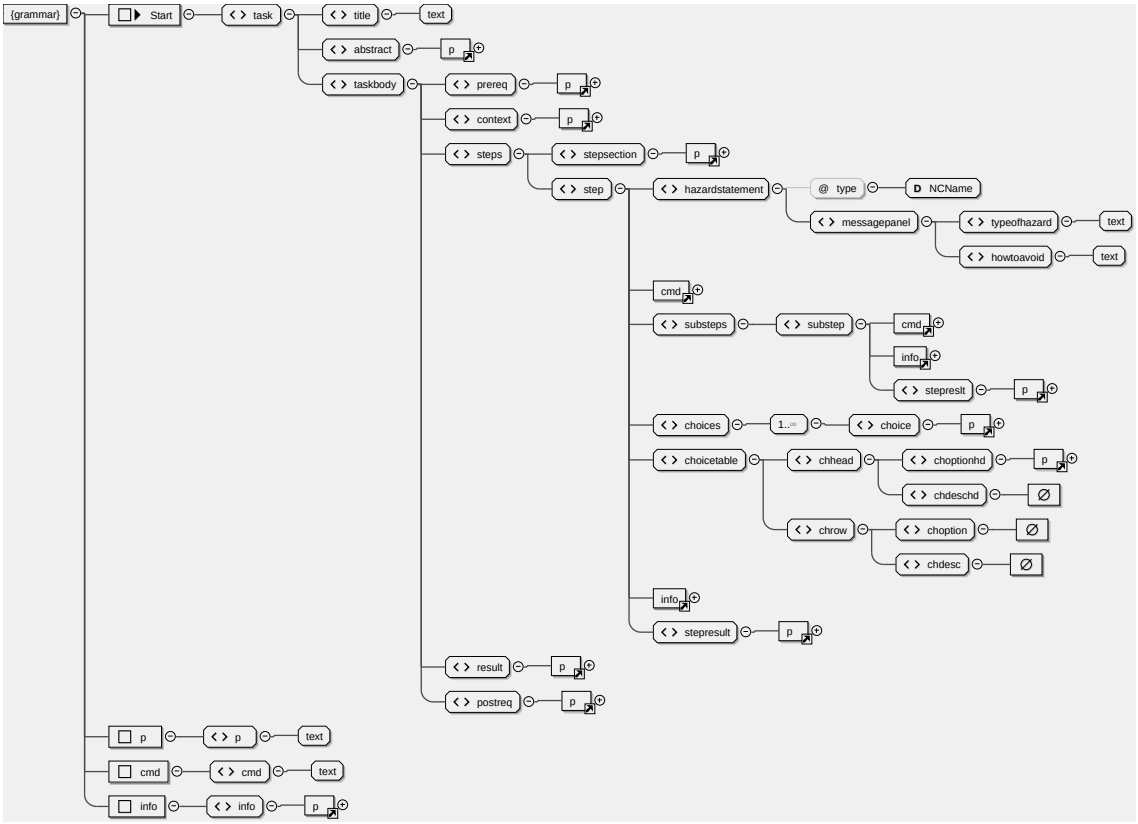
Das Element *task* beschreibt eine Handlungsanweisung oder Prozedur. Da es sehr komplex ist, wird dafür eine separate *section* in diesem Buch reserviert.

Ein gut ausgefüllter Task mit jeweils nur einem Kindelement ist im folgenden als XML Schnippsel dargestellt. Natürlich sind in den Elementen an bestimmten Stellen noch weitere Blockelemente erlaubt, wie Warnhinweise, Listen, Tabellen, etc.

```
<task>
    <title>Ein Task</title>
    <abstract>
        <p>Abstract-Elementin Task</p>
    </abstract>
    <taskbody>
        <prereq>
            <p>Prereq-Elementin Task</p>
        </prereq>
        <context>
            <p>Context-Elementin Task</p>
        </context>
        <steps>
            <stepsection>
                <p>Stepsection-Elementin Task</p>
            </stepsection>
            <step>
                <cmd>Cmd-Elementin Step</cmd>
                <substeps>
                    <substep>
                        <cmd>Substep-Elementin Step</cmd>
                        <info>
                            <p>Info-Elementin Substep</p>
                        </info>
                        <stepresult>
                            <p>Stepresult-Elementin Substep</p>
                        </stepresult>
                    </substep>
                </substeps>
                <choices>
                    <choice>
                        <p>Choice-Elementin Step</p>
                    </choice>
                    <choice>
                        <p>Zweites Choice-Elementin Step</p>
                    </choice>
                </choices>
                <choicetable>
                    <chhead>
                        <choptionhd>
                            <p>Choicetable mit leeren Zellenin Step</p>
                        </choptionhd>
                        <chdeschd/>
                    </chhead>
                    <chrow>
                        <choption/>
                        <chdesc/>
                    </chrow>
                </choicetable>
                <info>
                    <p>Info-Elementin Step</p>
                </info>
                <stepresult>
                    <p>Stepresult-Elementin Step</p>
                </stepresult>
            </step>
        </steps>
        <result>
            <p>Result-Elementin Task</p>
        </result>
        <postreq>
            <p>Postreq-Elementin task</p>
        </postreq>
    </taskbody>
```

```
</task>
```

Aus diesem XML Schnippsel kann man sich leicht die zugrundeliegende Grammatik als Grafik erzeugen lassen, z.B. mit dem oXygen XML Editor. (Da es sich um eine SVG Grafik handelt, sollte man auf einem Rechner verlustfrei reinzoomen können)



Buchwelt	DITA
Einleitung	Das *<abstract>* Element kann weitere Blockelemente beinhalten und dient zur initialen Beschreibung der Handlungsanweisung, Prozesses oder der Prozedur (Ich hoffe, dass ich alle in diesem Kontext relevanten Synonyme aufgezählt habe)
Vorbedingung	Im Rumpf der Task *taskbody* können Vorbedingungen angegeben werden, die erfüllt sein müssen bevor die Handlungsanweisung ausgeführt werden kann. In diesem *prereq* Element sind wieder weitere Blockelemente, wie Listen und Tabellen erlaubt. Natürlich sollte in der Ausgabe dieser Abschnitt irgendwie gekennzeichnet sein - so wie auch alle anderen Elemente - um diesen vom Fließtext zu unterscheiden.
Kontext	Im *<context>* Element wird beschrieben in welchem Kontext (Umgebung, Zweck und Nutzen) die Handlungsprozedur ausgeführt werden soll. Hier sind auch wieder viele Blockelemente erlaubt.
Inhalt der Arbeitsschritte	Schlussendlich hat man auch noch im *<steps>* Element die Möglichkeit weiteren Kapitelinhalt einzugeben - über das *<stepsection>* Element.

Arbeitsschritt	Das <step> Element selbst hat im einfachsten Fall ein <substeps> Element, so dass eine einfache verschachtelte nummerierte Liste abgebildet werden kann.
	
	`<step>`
	`  <cmd>Cmd-Elementin Step</cmd>`
	`  <substeps>`
	`  <substep>`
	`  <cmd>Substep-Elementin Step</cmd>`
	`  <info>`
	`  <p>Info-Elementin Substep</p>`
	`  </info>`
	`  <stepresult>`
	`  <p>Stepresult-Elementin Substep</p>`
	`  </stepresult>`
	`  </substep>`
	`  </substeps>`
	`  [...]`
	Im <cmd> Element steht dabei die auszuführende Aktion, während andere Elemente die Aktion weiter präzisieren. Besonders interessant ist dabei, dass z.B. Warnhinweise (<hazardstatement>) immer vor dem <cmd> Element bedatet werden müssen. Im <info> Element können zusätzliche Informationen stehen. Der <stepresult> dokumentiert schließlich das Ergebnis des Arbeitsschritts.

Auswahl der Aktionen	Hat der Ausführende des Handlungsschritts mehrere Optionen für die Aktion, dann kann das mittels des Elements <choices> modelliert werden. Statt einer Liste steht auch eine tabellarische Darstellung (<choicetable>) zur Verfügung, die neben der Aktion (<choption>) auch noch eine Beschreibung (<chdesc>) derselben erlaubt.
	
	`<choices>`
	`  <choice>`
	`  <p>Choice-Elementin Step</p>`
	`  </choice>`
	`  <choice>`
	`  <p>Zweites Choice-Elementin Step</p>`
	`  </choice>`
	`</choices>`
	`<choicetable>`
	`  <chhead>`
	`  <choptionhd>`
	`  <p>Choicetable mit leeren Zellenin Step</p>`
	`  </choptionhd>`
	`  <chdeschd/>`
	`  </chhead>`
	`  <chrow>`
	`  <choption/>`
	`  <chdesc/>`
	`  </chrow>`
	`</choicetable>`

Ergebnis der Handlungsanweisung	Werden alle Arbeitsschritte ausgeführt, dann sollte ein Ergebnis (<result>) dokumentiert werden. Diese Ergebnis-Elemente stehen auch im <step> und <substep> Element als <stepresult> Element zur Verfügung.
Nachbedingung	Analog zur Vorbedingung der Task kann auch eine Nachbedingung über das Element <postreq> dokumentiert werden.

3 Ausgewählte Themen

Contents

Auf den folgenden Seiten habe ich Themen ausgewählt, die für mich gerade besonders interessant erscheinen. Nach einer drei-jährigen Pause im Bereich XML, gibt es nun doch wieder viele neue Sachen ...

3.1 Transformationen mit XSLT

XSLT ist die Standardlösung für XML Transformationen. Es gibt zwar einige exotische Lösungen, wie:

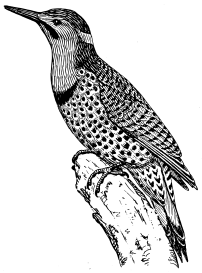
- Metamorphosis [61] ist eine proprietäre Sprache der Firma Ovidius GmbH in Berlin. Sie findet hauptsächlich Anwendung im Bereich Publishing in der Luftfahrt / Verteidigung.

61) https://de.wikipedia.org/wiki/XSL_Transformation#MetaMorphosis

- Es gibt auch auch Spielereien, wie eine Nachbildung der XSLT Syntax in Erlang: xmerl_xs [62)]

In diese Kapitel werden einige ausgewählte Themen zur XML Prozessierung mit XSLT dargestellt. Dabei geht es weder um Vollständigkeit, noch um die beste/eleganteste Lösung, sondern eher um die Vorstellung eines Anwendungsszenarios mit einem potentiellen Lösungsansatz - so wie ich die Sache halt angehen würde ...

3.1.1 Vortransformationen



Bei einer komplexen Transformation ist es ratsam und sogar manchmal unabdingbar die Konvertierung in einzelne Stufen aufzuteilen. Das hat folgende Vorteile:

- Der Prozess ist transparenter, da die einzelnen Stufen leichter überschaubar sind.
- Die Zwischenergebnisse können für Debug-Zwecke ausgewertet werde oder dienen als Eingabe für andere Prozesse.
- Nicht-relevante oder invalide Teilbäume können aus der Eingabeinstanz gefiltert werden, um so die weitere Verarbeitung zu beschleunigen.
- Hilfskonstrukte können erzeugt werden. Diese erleichtern die weitere Verarbeitung.

Es gibt zwei Möglichkeiten, wie eine Vortransformation eingebunden werden kann:

- In einem separaten File bzw. einer XML Instanz, die vom XSLT Prozessor vor der eigentlichen Transformation aufgerufen wird und einen Zwischenstand produziert. Dieser kann dann als Eingabe für den Haupttransformationsschritt dienen.
- Innerhalb des eigentlichen XSLT Stylesheets. Hier wird das Ergebnis der Vortransformation in einer Variablen erzeugt.

Den zweiten Punkt möchte ich anhand eines Beispiel XSLT Skripts vorführen. Betrachten wir folgende Input Daten:

```
<education-system>
  <administrative-regions>
    [...]
    <dministrative-region id="31" name="Bavaria">
      <shools>
        <school id="45">
          <teachers>
            <teacher id="576"/>
            <teacher id="345"/>
            <teacher id="12"/>
          </teachers>
        </school>
        <school id="36">
          <teachers>
            <teacher id="576"/>
            <teacher id="8"/>
          </teachers>
        </school>
        [...]
      </shools>
    </dministrative-region>
    [...]
  </administrative-regions>
```

62) http://erlang.org/doc/man/xmerl_xs.html

```
</education-system>
```

Die erste Datei beinhaltet eine Zuordnung von Lehrern zu Schulen in verschiedenen Regierungsbezirken. Um die Daten zu den beiden referenzierten Objekten einzusehen, müssen zwei weitere Dateien konsultiert werden. Die Datei, welche die Lehrer auflistet:

```
<teachers>
  [...]
  <teacher id="576">
    <first-name>Alfons</first-name>
    <last-name>Blimetsrieder</last-name>
    <subjects>
      <subject>Biology</subject>
      <subject>Math</subject>
      <subject>Sport</subject>
    </subjects>
    <suspended>2017-12-31</suspended>
    [...]
  </teacher>
  [...]
</teachers>
```

Und die Datei, welche die Schulen auflistet:

```
<schools>
  [...]
  <school id="45">
    <name>Gymnasium Bad Aibling</name>
    <type>Oberschule</type>
    [...]
  </school>
  [...]
</schools>
```

Um diese Daten verarbeiten zu können ist es sinnvoll, die drei Dateien in einem ersten "Resolver" Schritt zusammenzuführen und ggf. irrelevante Strukturen zu entfernen. Lehrer aus obigem Beispiel können beispielsweise suspendiert worden sein. Das folgende Skript erledigt dies mittels einer zusätzlichen Transformation in eine Variable:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet version="2.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  exclude-result-prefixes="#all">

  <xsl:output indent="yes" method="xml"/>

  <xsl:strip-space elements="*"/>

  <xsl:param name="file-1" required="yes"/>
  <xsl:param name="file-2" required="yes"/>
  <xsl:param name="file-3" required="yes"/>

  <xsl:variable name="files" select="(doc($file-1), doc($file-2), doc($file-3))"/>
  <xsl:variable name="bavaria-region-ids" select="(31, 58)"/>

  <xsl:key name="teachers" match="teacher" use="@id"/>
  <xsl:key name="schools" match="school" use="@id"/>

  <xsl:template name="main">
    <xsl:variable name="resolve-result">
```

```
      <xsl:apply-templates select="$files/administrative-regions" mode="resolve"/>
    </xsl:variable>
    <xsl:apply-templates select="$resolve-result/administrative-regions"/>
  </xsl:template>

  <xsl:template match="administrative-region[not(@id = $bavaria-region-ids)]"
                mode="resolve"/>

  <xsl:template match="school" mode="resolve">
    <xsl:copy>
      <xsl:copy-of select="key('schools',@id, $files/schools[1]/root())/node()"/>
      <xsl:apply-templates select="node()|@*" mode="resolve"/>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="teacher" mode="resolve">
    <xsl:copy-of select="key('teachers',@id, $files/teachers[1]/root())/node()"/>
  </xsl:template>

  <xsl:template match="teacher[suspended/xs:date(.) le current-date()]"/>

  <xsl:template match="node()|@*" mode="#all">
    <xsl:copy>
      <xsl:apply-templates mode="#current"/>
    </xsl:copy>
  </xsl:template>

</xsl:stylesheet>
```

Im ersten Resolve-Schritt werden die Referenzen zu den Lehrer- und Schul-Objekten aufgelöst, d.h. die Attribute des Schul-Objekts werden in die Struktur aus der ersten Datei kopiert.

Die Liste der Lehrer an diesen Schul-Objekten bleibt erhalten und wird mit dem Inhalt aus der zweiten Datei bestückt.

Zusätzlich werden alle Regierungsbezirke entfernt, die nicht zu Bayern gehören - was die weitere Verarbeitung wesentlich beschleunigen wird. Lehrer die suspendiert worden sind fliegen ebenfalls raus ...

In-Situ Vortransformation

Als ich mir kürzlich meinen Code vor zehn Jahren zu Gemüte führte, fiel mir ein sehr seltsames Stück XSLT auf, sinngemäss:

```
Datei: common/semantic-tables.xsl

<xsl:template name="maintenance-table">
    <cals-table-structure>
        [...]
            <<xsl:apply-templates select="maint-int"/>
        [...]
    </cals-table-structure>
</xsl:template>

<xsl:template match="maintenance-table">
    <xsl:variable name="maintenance-table">
        <xsl:call-template name="maintenance-table"/>
    </xsl:variable>
    <xsl:apply-templates select="$maintenance-table"/>
</xsl:template>
```

Was hat mich denn da geritten? Ich bin ich dann schon darauf gekommen... Der Clou ist hier eine Vortransformation innerhalb einer Match-Regel.

Die Transformation innerhalb der Variablen rendert die semantischen Elemente der Wartungstablle, wie z.B. das Wartungsintervall `maint-int` , in eine CALS-Tabelle.

Diese wird dann im nächsten Transformationsschritt entweder nach XSL-FO oder nach HTML transformiert, je nachdem welche Haupt-Datei `main.xsl` das Modul `semantic-tables.xsl` importiert.

```
Datei html/main.xsl

<xsl:import href="common/semantic-tables.xsl"/>

<xsl:template match="cals-table-structure">
    <html-tabllen-struktur>
        [...]
    </html-tabllen-struktur>
</xsl:template>

Datei pdf/main.xsl

<xsl:import href="common/semantic-tables.xsl"/>

<xsl:template match="cals-table-structure">
    <pdf-tabllen-struktur>
        [...]
    </pdf-tabllen-struktur>
</xsl:template>
```

Dieser Ansatz ist sehr flexibel, denn im herausfaktorisierten Tabellenalgorithmus können leicht Sonderfälle, wie Duplikat-Eliminierung oder spezielle Merge-Operation, abgefangen werden. Zudem können sowohl das Named-Template bzgl. der Wartungstabelle als auch die Match-Regel im importierenden Stylesheet überschrieben werden, was der Kreativität keine Grenzen setzt.

Mehrstufige Transformationen

Bei manchen Kovertierungen reichen ein oder zwei hintereinander geschaltete Transformationsschritte nicht aus.

Vielleicht hat man es mit einer Struktur zu tun, die gar nicht zum Zielformat passt... Sei es, weil auf ein sehr restriktives Inhaltsmodel transformiert wird, sei es weil ERROR! Linktarget does not exist erst schrittweise erschlossen wird, da der Überblick über die Daten noch fehlt.

In jedem Fall muss für manche Elemente in einem späteren Schritt entschieden werden, wohin sie umsortiert werden sollten, weil sie jetzt gerade stören.

Beispiele:

1. Transformationen auf ein restriktiveres Inhaltsmodell
   - Das XHTML eines Webeditors, bei dem der User willkürlich die unterschiedlichsten Strukturen eingeben kann, wird auf die restrikte DTD eines XML Redaktionssystems gemappt.
2. Ein restriktives, strukturiertes Inhaltsmodell soll in ein freieres, strukturiertes Inhaltsmodell konvertiert werden:
   - Diese Form der Transformation findet man oft bei den Ausgabestrecken eines XML Redaktionssystems. Hier kommt man gut mit 2-3 Transformationsschritten aus. Diese werden dazu verwendet, um die Konvertierung zu erleichtern, und nicht aus "Platzgründen" wie in Punkt 1.)

3. Transformationen von relativ unstrukturierten Daten. Hier muss man die Strukturen erst bilden. Das kann durch Einsammeln gleichartiger Elemente passieren, die dann hierarchisch in einer Baumstruktur geordnet werden.
   - Baumstrukturen kann man z.B. auch aus Dateipfaden generieren, die dann mittels anderer Datenquellen noch angereichert werden, bspw. CSV-Dateien.

Mehrstufige Transformationen werden in ERROR! Linktarget does not exist 9a77a304-db82-437f-95d3-02656d93692f eingesetzt.

3.1.2 Komplexe XML-2-XML Transformationen

Der erfahrene XML-Entwickler schreibt schlanken, performanten und einfachen Code, den auch andere gut verstehen. Er meistert alle Bereiche der XML-Entwicklung und hat sich mit Publishing Standards befasst, wie Docbook, DITA und JATS. Er erstellt mittels XSL-FO und verschiedenen Seitenvorlagen schöne PDF Dokumente und hat auch schon in anderen Anwendungsbereichen geabeitet, wie bspw. im EDI[63] Umfeld. Er beherrscht XML Datenbanken, wie Marklogic oder eXist und deren Abfragesprache XQuery.

Als Königsdisziplin stellen sich komplexe XML-2-XML Transformationen heraus. Insbesondere solche, die von einem relativ freien Inhaltsmodell auf ein restriktives Inhaltsmodell abbilden. Dabei können diese ausserhalb und auch innerhalb einer XML Datenbank ablaufen - oder über Webrequests verteilt statt finden.

Es hat sich bewährt solche komplexen Transformationen auf mehrere Schritte aufzuteilen, die jeder für sich genommen, eine abgeschlossene und leicht zu testende Einheit bildet.

Schritt-für-Schritt wird dabei die XML Eingabeinstanz transformiert, bis schliesslich das validierbare Ergebnis herauskommt. Das XML der Zwischenschritte kann dabei meistens nicht gegen ein Schema validiert werden, weshalb eine besondere Sorgfalt bei der Entwicklung erforderlich ist.

Schritt-für-Schritt Python Skript

Bei einer mehrstufigen Transformation möchte man bei der Entwicklung leicht die Zwischenschritte überprüfen können. Dabei hilft eine andere Skriptsprache, wie bspw. Python. Das folgende Skript nimmt die XML Daten in einem Ordner `input`, transformiert diese in Sortierreihenfolge mit den XSLT Skripten, die im Ordner `xslt` liegen und schreibt die Ausgabe übersichtlich in den Ordner `output`.

```
import glob, os, shutil, getopt, sys, subprocess

SAXON_JAR = "/mnt/c/saxon/saxon9pe_98.jar"
JAVA_CMD = "java"

def transform():
    for fpathin glob.glob('input/*'):
        file_name= os.path.basename(fpath)
        input_folder = os.path.dirname(os.path.realpath(fpath))
        input_file =  os.path.join(input_folder, file_name)

        steps = os.listdir("xslt")
        steps.sort()
        step = None

        for step_filein steps:
            if not step_file.startswith("step"): continue
            step = step_file.split("_")[0]
            output_folder = input_folder.replace("input","output/"+step)
            current_step = os.path.join(output_folder, file_name)
```

63) https://de.wikipedia.org/wiki/Elektronischer_Datenaustausch

```
            os.makedirs(output_folder,exist_ok=True)
            args = [
                JAVA_CMD,
                "-classpath",
                SAXON_JAR,
                "net.sf.saxon.Transform",
                "-s:"+input_file,
                "-o:"+current_step,
                "-xsl:xslt/"+step_file,
                "filename="+os.path.basename(input_file).replace(".xml","")
                ]
            try:
                subprocess.call(args)
            except:
                print ("ERROR: Could not transform file: "+fpath+" with: "+step_file)
            input_file = current_step
            print ("Transformed "+step+": "+fpath)

transform()
print ("Done")
```

Das Skript kann natürlich noch um weitere Funktionen erweitert werden, wie bspw. einer Validierung für den letzten Schritt oder einem Deltavergleich der Zwischenergebnisse mit denen des vorherigen Transformationslaufs.

Will man das Ganze noch weiter treiben, kann man auch eine BPMN Engine [64], wie Camunda[65] verwenden (einen speziellen Task-Executor für Camunda, der genau für diese XML Zwecke gemacht wurde, findet man auch in meinem Github Repository[66]).

Patterns für wiederkehrende Schritte

Eine mehrstufige Transformation, die auf ein restriktives Inhaltsmodell abbildet, funktioniert vielleicht wie eine Goldschürfer-Pipeline, in der gesiebt und gerüttelt wird, bis das erwartete Ergebnis vorliegt.

Folgende Patterns für wiederkehrende Schritte lassen sich dabei identifizieren:

Elemente markieren

Wenn man alles auf einmal transformieren will, kommt man schnell in Bedrängnis. Es empfiehlt sich zunächst zu markieren und im nächsten Schritt dann auf den markierten Elementen bestimmte Operationen auszuführen.

```
<xsl:template match="*[name()=$outline-element-names]">
  <xsl:copy>
    <xsl:apply-templates select="@*"/>
    <xsl:if test="preceding-sibling::*[1][@otherprops=$list-fragment-marker]">
      <xsl:attribute name="copy-target-id"
                     select="preceding-sibling::*[@otherprops=$list-fragment-marker][1]/
                             descendant::*[@copy-id][last()]/@copy-id"/>
    </xsl:if>
    <xsl:apply-templates select="node()|@*"/>
  </xsl:copy>
</xsl:template>
```

Hier wird ein künstliches Attribut `@copy-target-id` mit einem Wert von `@copy-id` an ein Outline Element gesetzt, das im folgenden Schritt an die Stelle nach der `@copy-id` kopiert wird.

64) https://de.wikipedia.org/wiki/Prozessmanagement
65) https://camunda.com/
66) https://github.com/alexdd/tektur_worker

Elemente kopieren

Ein wiederverwendbarer Schritt, der mit `@copy-target-id` markierte Elemente nach eine Stelle kopiert, die mit `@copy-id` markiert wurde, könnte z.B. so aussehen:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    exclude-result-prefixes="xs"
    version="2.0">

    <xsl:key name="targets" match="*[@copy-target-id]" use="@copy-target-id"/>

    <!-- copy elements from src to target -->

    <xsl:template match="*[@copy-id]">
        <xsl:copy>
            <xsl:apply-templates select="node()|@*"/>
            <xsl:apply-templates select="key('targets',@copy-id)" mode="copied"/>
        </xsl:copy>
    </xsl:template>

    <!-- remove original position and attributes -->

    <xsl:template match="*[@copy-target-id]"/>
    <xsl:template match="@copy-target-id|@copy-id" mode="copied"/>

    <xsl:template match="node()|@*" mode="#all">
        <xsl:copy>
            <xsl:apply-templates select="node()|@*" mode="#current"/>
        </xsl:copy>
    </xsl:template>

</xsl:stylesheet>
```

Mit so einer Vorgehensweise kann man sukkzessive und mittels einzelner Kopierschritte die XML Instanz umbauen und die Zwischenergebnisse verfolgen. So eine explorative Herangehensweise hat enorme Vorteile, wenn man sich über den Algorithmus noch nicht ganz im Klaren ist.

Elemente nach oben ziehen

Falls ein tieferliegendes Element nicht an die Stelle in der Ziel-DTD passt, kann man es mit folgenden Templates "nach oben ziehen":

```
<xsl:template match="table[descendant::table or descendant::ol]">
  <xsl:copy>
    <xsl:apply-templates mode="remove-table-ol"/>
  </xsl:copy>
  <xsl:apply-templates select="descendant::ol | descendant::table"/>
</xsl:template>

<xsl:template match="*[self::ol or self::table]" mode="remove-table-ol"/>
```

Hier werden Tabellen und Listen in einer Tabelle nach der Tabelle gesetzt. Über einen Modus (vgl. auch ein Beispiel zum Modus hier: Modus vs. Tunnel Lösung on page 66 werden diese Knoten aus dem XML Zielbaum "ausgeschnitten".

Blöcke auszeichnen

Falls Blockstrukturen geklammert werden sollen, bspw. wenn diese im HTML Kapitel nur mittels `h1` Überschriften-Tags gekennzeichnet sind, dann hilft vielleicht ein Template wie dieses weiter:

```
<xsl:template match="body">
  <xsl:for-each select="h1">
```

```
    <block>
      <title>
        <xsl:apply-templates />
      </title>
      <xsl:apply-templates select="following-sibling::*[not(self::h1)]
          [preceding-sibling::h1[1][generate-id()=current()/generate-id()]]"/>
    </block>
  </xsl:for-each>
</xsl:template>
```

Mittels der XPath `fn:is()` Funktion liesse sich der `generate-id()` Vergleich sogar noch abkürzen.

Mixed Content wrappen

Sehr unangenehm ist sporadisch auftretender XML Mixed Content, z.B. zwischen Paras. Mit folgenden Templates lässt sich das handeln. Zuerst kann man den Mixed Content in einem vorhergehenden Schritt markieren und in einen künstlichen Para packen ...

```
<!-- wrap a p around PCDATAin li -->

<xsl:template match="text()[parent::li]">
  <p content="mixed">
    <xsl:value-of select="."/>
  </p>
</xsl:template>
```

... hier markiert mit `@content="mixed"` . Im folgenden Schritt werden dann die ursprünglichen Paras, die jetzt verschachtelt im künstlichen Para liegen wieder ausgepackt:

```
<xsl:variable name="inline-elements" select="('sub','sup','b','i','br','u')"/>

<xsl:template match="p[@content='mixed' and not(preceding-sibling::p[@content='mixed'])]">
  <xsl:variable name="first-p-id" select="(preceding-sibling::*[1]/generate-id(),
                                                                  generate-id())[1]"/>
  <p>
    <xsl:copy-of select="preceding-sibling::*[name()=$inline-elements]"/>
      <xsl:apply-templates select="node()|following-sibling::*[(self::p[@content='mixed']
                          or name()=$inline-elements) and
      not(preceding-sibling::p[not(@content='mixed')][1]/
                          generate-id()!=$first-p-id)]"mode="unwrap"/>
   </p>
</xsl:template>

<xsl:template match="p[@content='mixed' and preceding-sibling::p[@content='mixed']]"/>

<xsl:template match="p" mode="unwrap">
  <xsl:apply-templates/>
</xsl:template>

<xsl:template match="*[not(self::p)]" mode="unwrap">
  <xsl:copy>
    <xsl:apply-templates select="node()|@*"/>
  </xsl:copy>
</xsl:template>

<xsl:template match="li[p[@content='mixed']]/*[name()=$inline-elements]"/>
```

3.1.2.1 Vererbung

Mit XSLT kann man Konstrukte aus anderen Programmierparadigmen nachbilden, bspw. Vererbung. Dabei wird in einer Spezialisierung eine schon bereits getätigte Implementierung übernommen und erweitert oder eingeschränkt.

Der Vorteil dabei ist, dass man nicht alles nochmal neu schreiben muss. Das verkleinert die Redundanz, führt zu einer besseren Wartbarkeit und einer geringeren Fehleranfälligkeit.

Beispiel: Parameterisierung

Gewöhnlich implementiert man ein Stylesheet für ein bestimmtes Ausgabeformat und eine Produktvariante. Schrittweise werden dann weitere Varianten und Formate hinzugefügt.

Am komfortabelsten hat man es natürlich, wenn zu Beginn der Implementierung eine vollständige Spezifikation vorliegt... Das ist aber natürlich eher selten der Fall.

Aus diesem Grund ist es wichtig, sich eine gute Strategie zu überlegen, damit die Architektur nicht in Spaghetthi-Code ausartet.

Eine gute Option wäre, die XSLT Import Präzedenz auszunutzen, vgl. Kapitel Eindeutigkeit der Regelbasis on page 21.

Will man zu einem späteren Zeitpunkt weitere Parameter einzuführen, dann müsste ein Switch, wie der folgende, an mehreren Stellen im Code aktualisiert werden.

```
<xsl:choose>
    <xsl:when test="$myParameter='this_option'">
        <!-- do this -->
    </xsl:when>
    <xsl:when test="$myParameter='that_option'">
        <!-- do that -->
    </xsl:when>
        [...]
</xsl:choose>
```

Besser ist es, wenn man ein Core-Stylesheet pflegt, das für ein Format und eine Produktvariante gut ausgetestet ist. Dieses Core-Stylesheet wird dann einfach für eine neue Variante importiert und relevante Teile werden für die neue "Spezialisierung" überschrieben. Beispielsweise könnte eine Regel zum Setzen des Headers auf jeder Seite so implementiert sein:

```
<xsl:template name="render-header">
    <!-- print logo on the left side spanning two rows-->
    <!-- print some metadata right side first row -->
    <!-- print a running header right side second row -->
</xsl:template>
```

Wird in einem neuen Format, bspw. A5, diese Logik ausgetauscht und nur eine Zeile gedruckt, z.B. weil man nicht so viel Platz hat, so würde in einem "abgeleiteten" Stylesheet einfach die Regel noch einmal implementiert.

```
<xsl:choose>
<xsl:template name="render-header">
    <!-- print a running header on left side -->
    <!-- print logo on right side -->
</xsl:template>
```

Dieses Template hat nun Vorrang und wird zur Auswertung herangezogen, mit der Konsequenz, dass der Header nur einzeilig gedruckt wird. Das schöne an diesen "Named-Templates" ist auch, dass man sie innerhalb von Variablen verwenden kann:

```
<xsl:variable name="margin-width">
    <xsl:call-template name="get-margin-width"/>
</xsl:variable>
```

Das Template `get-margin-width` kann in einem "Sub"-Stylesheet überschrieben werden, ohne dass die Variablen-Zugriffe im Core-Stylesheet angepasst werden müssen. Eine Zuweisung, wie:

```
width="{$margin-width}"
```

müsste nirgendwo im Code nochmal angefasst werden, was natürlich sehr komfortabel ist.

Beispiel: Spezialisierte Fallunterscheidung

Kommt ein Ausgabeformat zu einem späteren Zeitpunkt hinzu, so kann man die Vererbung nutzen um clever zwischen den Ausgabeformaten zu unterscheiden. Ein bestehendes Template `"cover"` :

```
<xsl:template match="cover">
  <fo:page-sequence master-reference="coverFront" force-page-count="auto">
    <fo:flow flow-name="xsl-region-body">
    [...]
  <fo:page-sequence master-reference="coverBack" force-page-count="auto">
    <fo:flow flow-name="xsl-region-body">
    [...]
```

kann aufgesplittet werden in ein Template für eine Vor- und Rückseite:

```
<xsl:template match="cover">
  <fo:page-sequence master-reference="coverFront" force-page-count="auto">
    <fo:flow flow-name="xsl-region-body">
      [...]
</xsl:template>

<xsl:template name="cover-back-switch">
  <xsl:call-template name="cover-back"/>
</xsl:template>

<xsl:template name="cover-back">
  <fo:page-sequence master-reference="coverBack" force-page-count="auto">
    <fo:flow flow-name="xsl-region-body">
      [...]
```

Das Template `"cover-back-switch"` kann dann in einem Ausgabeformat überschrieben werden, um bspw. die Cover-Rückseite abhängig von einer Bedingung zu schalten.

Einen Algorithmus in einer Superklasse grob zu strukturieren und Details in Unterklassen zu verfeinern, ist ein gängiges Design Pattern in der Objektorientierten Programmierung. Diese Methodik kann auch sehr schön mittels Named-Templates in XSLT realisiert werden. Nicht umsonst heisst das OO Pattern dazu Template Method[67)]

67) https://en.wikipedia.org/wiki/Template_method_pattern

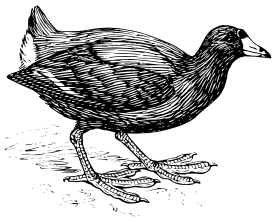
3.1.3 XSLT Streaming

Bei grossen, flach strukturierten Datenmengen gibt es zwei Möglichkeiten:

1. Für einfache Sammel- und Auswertungsaufgaben schreibt man sich am besten einen kleinen Parser, z.B. mit der Pythonsgmllib[68].
2. Für komplexere Aufgaben, in denen man nicht an jeder Stelle über den ganzen XML Baum navigiert und sich die Werte zusammensuchen suchen muss, kann man die Streaming Funktion des Saxon XSLT Prozessors verwenden.

XSLT Streaming ist in der XSLT Version 3.0[69] neu hinzugekommen und in der kommerziellen Saxon-EE Lösung von Michael Kay[70] implementiert. Bei dieser Methode wird kein Eingabebaum im Speicher aufgebaut, was zu einer drastischen Performanzsteigerung führt.

Es gibt ein paar Regeln, die man bei der Verarbeitung großer Datenmengen über die Streaming Funktionen beachten sollte:



- Bei einer XPATH Auswertung sollte nur ein einfacher Ausdruck mit höchstens einer konsumierenden Selektion gegeben sein. Konsumieren heißt, dass vom Kontextknoten aus eine Knotenmenge abwärts selektiert wird. Dagegen bleibt die Information bzgl. der Ancestor-Achse erhalten.
- Bei einer Selektion sollte man aber darauf achten nur atomare Werte auszuwählen.
- Knotenmengen, die über die Streaming Option eingelesen wurden, können nicht einer Funktion übergeben werden. Sie sind auch nicht einer Variablen zuweisbar.
- "Crawler"-Ausdrücke, wie `//section` sind nicht nicht zu verwenden, ebenso ein rekursiver Abstieg mit Selektion, wie bspw. mit einem `apply-templates` Call.

Zu Beginn der Streaming-Aktion kann man sich auf konventionelle Art und Weise Teilbäume, die nicht so performanzlastig aufgebaut werden, in einer Variablen abspeichern und im Verlauf der Streaming-Verarbeitung z.B. für einen Vergleich auswerten.

3.1.3.1 XSLT Akkumulator

Ein einfaches Streaming Stylesheet könnte z.B. so aussehen:

```
<xsl:stylesheet version="3.0"
  xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  exclude-result-prefixes="#all">

  <xsl:output method="xml" indent="yes"/>

  <xsl:mode on-no-match="shallow-copy" use-accumulators="entry-count" streamable="true"/>

  <xsl:accumulator name="entry-count" as="xs:integer" initial-value="0"
                   streamable="yes">
    <xsl:accumulator-rule match="entry" select="$value + 1"/>
  </xsl:accumulator>

  <xsl:template match="/">
    <result>
      <xsl:apply-templates/>
      <count>
```

68) https://docs.python.org/2/library/sgmllib.html
69) https://www.saxonica.com/html/documentation/sourcedocs/streaming/xslt-streaming.html
70) https://www.saxonica.com/html/documentation/sourcedocs/streaming/

```
        <xsl:value-of select="accumulator-after('entry-count')"/>
      </count>
    </result>
  </xsl:template>

</xsl:stylesheet>
```

Diese Stylesheet hat einige Besonderheiten:

Zum einen wird darin ein Default-Modus deklariert, der jeden Knoten der Eingabeinstanz über eine implizite Identity-Transformation (shallow-copy)[71] in die Ausgabeinstanz kopiert.

(Auf herkömmlichem Weg würde man dafür ein Templates, wie dieses, verwenden:)

```
<xsl:template match="node()|@*"
  <xsl:copy>
    <xsl:apply-templates select="node()|@*"/>
  </xsl:copy>
</xsl:template>
```

Zum anderen wird ein Akkumulator verwendet.

Normalerweise gibt es in XSLT keine Variablen, sondern nur Konstanten, wie das auch bei funktionalen Programmiersprachen der Fall ist.

- Es gibt zwar schon länger eine Saxon-Erweiterung, die die mehrmalige Zuweisung eines Wertes an eine Variable erlaubte. Im Normallfall braucht man diese Eigenschaft aber nicht.

Ber der Verarbeitung sehr großer Datemengen, sind aber zuweisbare Variablen unumgänglich, denn sonst würde der Laufzeitstapel schnell an seine Grenzen gelangen.

Ein Akkumulator akkumuliert Werte. Das können atomare Typen sein, wie im obigen Beispiel, aber auch Datenstrukturen können aufgebaut werden, wie bspw. der gerade prozessierte Teilbaums in einem Dictionary.

Im Akkumulator muss das `streamable="yes"` Property gesetzt sein, wenn er im Streaming-Modus arbeiten soll. In diesem Modus kann der Akkumulatorwert erst ausgelesen werden, wenn der untersuchte Baum vollständig durchlaufen wurde.

Um die Unterschiede zum "normalen" XSLT Betrieb festzustellen, können im obigen Beispiel einige Änderungen vorgenommen werden, die der Streaming Prozessor nicht akzeptiert. Diese vermeintlich fehlerhafte Eingabe quittiert Saxon mit der folgenden Fehlermeldung:

71) https://www.saxonica.com/html/documentation/xsl-elements/mode.html

```
Cannot call accumulator-after except during the post-descent
phase of a streaming template
```

Diese Fehlermeldung erscheint, wenn man den ***apply-templates*** Call entfernt. Der Akkumulator wird also nur befüllt, wenn der Baum auch explizit durchlaufen wurde.

Dieser Durchlauf kann auch ein reines Kopieren sein, bspw. kann man den `apply-templates` Call auch durch ein...

```
<xsl:copy-of select="."/>
```

... ersetzen, was gleichbedeutend mit der Mode Einstellung

```
on-no-match="deep-copy"
```

wäre. Wie man sieht hat sich in XSLT 3.0 viel bzgl. der Handhabung verschiedener Verarbeitungsmodi getan. Anstatt Default-Match Regeln zu schreiben, setzt man in der Stylesheet Deklaration bestimmte Modus Properties, die den Baumdurchlauf auf verschiedene Arten realisieren.

Die Verarbeitung großer Datenmengen ist aber mit Streaming etwas tricky!

- Es sollte immer geprüft werden, ob auch ein ggf. konventionelles Performanz-optimiertes XSLT für den Anwendungsfall ausreicht.

3.1.3.2 XSLT Iterator

Betrachten wir folgendes Problem. Es soll ein kommaseparierter Report aus dieser XML Quelle generiert werden.

```
<status-report>
  <status-change>
    <billing_id>360788</billing_id>
    <claim_ids>967382,673647</claim_ids>
    <status>open</status>
    <time_stamp>2019-02-22T13:53:34.605Z</status_time>
  </status-change>
  <status-change>
    <billing_id>360788</billing_id>
    <claim_ids>967382,673647</claim_ids>
    <status>open</status>
    <time_stamp>2019-02-22T13:53:34.605Z</status_time>
  </status-change>
  [...]
```

Mit einer `for-each` Loop und einem *Named-Template* würde das so gehen:

```
<xsl:template name="main">
  <xsl:for-each select="$input-file/status-report/status-change">
    <xsl:value-of select="concat(billing_id,',')"/>
    <xsl:value-of select="concat(claim_ids,',')"/>
    <xsl:value-of select="concat(status,',')"/>
    <xsl:value-of select="concat(format-dateTime(xs:dateTime(time_stamp),
                                 '[Y]-[M]-[D] [H]:[m]'),'&#10;')"/>
  </xsl:for-each>
</xsl:template>
```

▶ Named-Templates, die direkt über den Saxon Aufruf `saxon -it:main` aufgerufen werden, sind dann brauchbar, wenn keine eindeutige Eingabequelle vorhanden ist, bspw. weil aus mehreren Quellen eingelesen werden soll, falls die Eingabe von einem Webservice kommt oder vom XSLT Skript selbst erzeugt wird.

Diesen Code kann man vereinfachen: Da von einer Datei eingelesen wird, brauchen wir kein *Named-Template*. Statt der Schleife können wir uns auch auf den rekursiven Abstieg des XSLT Prozessors verlassen:

```
<xsl:template match="/status-report/status-change">
  <xsl:value-of select="concat(billing_id,',')"/>
  <xsl:value-of select="concat(claim_ids,',')"/>
  <xsl:value-of select="concat(status,',')"/>
  <xsl:value-of select="concat(format-dateTime(xs:dateTime(time_stamp),
                               '[Y]-[M]-[D] [H]:[m]'),'&#10;')"/>
</xsl:template>
```

Wollen wir große Datenmengen schnell verarbeiten - mit ein paar Hundert MB, so ist es sinnvoll auf die neue XSLT3.0 Streaming Option umzuschalten, weil dadurch kein Eingabebaum in-Memory aufgebaut wird. Wie schon im Kapitel XSLT Akkumulator on page 60 angesprochen, gibt es dazu mehrere Möglichkeiten.

Wir betrachten hier das `xsl:iterator` (Doku)[72] Konstrukt und stossen dabei auf einige Fallstricke. Zunächst zu den

Settings:

72) https://www.saxonica.com/html/documentation/xsl-elements/iterate.html

- Wir benutzen `xsl:source-document` in Verbindung mit dem `streamable='yes'` Attribut, um dem Prozessor mitzuteilen, dass er im Streaming Modus arbeiten soll.
- Wenn wir die Quelle über einen Parameter einlesen, dann müssen wir auch die Transformation über ein Named-Template starten.

Ohne zu wissen, wie XSLT Streaming genau funktioniert, setzen wir probeweise eine Reihe von `value-of select` statements in den Iterator:

```
<xsl:template name="main">
  <xsl:source-document href="{$input-file}" streamable='yes'>
    <xsl:iterate select="status-report/status-change">
      <xsl:value-of select="concat(billing_id,',')"/>
      <xsl:value-of select="concat(claim_ids,',')"/>
      <xsl:value-of select="concat(status,',')"/>
      <xsl:value-of select="concat(format-dateTime(xs:dateTime(time_stamp),
                                   '[Y]-[M]-[D] [H]:[m]'),'&#10;')"/>
    </xsl:iterate>
  </xsl:source-document>
</xsl:template>
```

und werden dafür prompt mit einer Fehlermeldung belohnt:

```
Static error on line 16 column 64 of report.xsl:
  XTSE3430: The body of the xsl:stream instruction is not streamable
  *  There is more than one consuming operand: {xsl:value-of} on line 18, and
  {xsl:value-of} on line 19
```

In diesem Iterator ist also nur eine "konsumierende" `value-of` Operation erlaubt. Um nur einmal zu selektieren, müssen wir - auf Kosten der Lesbarkeit - ziemlich umbauen. Eine Lösung könnte z.B. so aussehen:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:xs="http://www.w3.org/2001/XMLSchema"
  exclude-result-prefixes="xs"
  xpath-default-namespace="https://tekturcms.de/schema/status-report/1.0"
  version="3.0">

  <xsl:param name="input-file" required="yes"/>

  <xsl:output method="text"/>

  <!-- https://www.saxonica.com/html/documentation/xsl-elements/iterate.html -->

  <xsl:template name="main">
    <xsl:source-document href="{$input-file}" streamable='yes'>
      <xsl:iterate select="status-report/status-change/*">
        <xsl:choose>
          <xsl:when test="name()='time_stamp'">
            <xsl:value-of select="concat(format-dateTime(xs:dateTime(time_stamp),
                                         '[Y]-[M]-[D] [H]:[m]'),'&#10;')"/>
          </xsl:when>
          <xsl:otherwise>
            <xsl:value-of select="concat(.,',')"/>
          </xsl:otherwise>
        </xsl:choose>
      </xsl:iterate>
    </xsl:source-document>
```

```
  </xsl:template>
</xsl:stylesheet>
```

Hier wird davon ausgegangen, dass das Element mit Namen 'time_stamp' als letztes in der Sequenz vorkommt und beim Auftreten (`
`) wird ein Zeilenumbruch gesetzt. Der deklarative Ansatz aus dem ersten Beispiel geht dabei verloren.

> ▶ Logisch gesehen wird beim XSLT Streaming auf einer niedrigeren Abstraktionsebene programmiert, um den Anforderungen des Prozessors gerecht zu werden.

Für eine **1.6 GB Datei** benötigt das obige Skript auf meinem Rechner gute **drei Minuten**. Der traditionelle template-match Ansatz bricht mit einer Out-of-Memory Exception ab, selbst wenn man den Java Heap Size auf 4GB einstellt.

3.1.4 Reguläre Ausdrücke

Manchmal reicht für das Durchsuchen des XML Baums XPATH nicht mehr aus und man will auf den Textknoten reguläre Ausdrücke aufrufen. Ich kenne drei XPATH Funktionen, in denen man reguläre Ausdrücke angeben kann:

- `fn:matches(subject, pattern, flags)`
- `fn:replace(subject, pattern, replacement, flags)`
- `fn:tokenize(subject, pattern, flags)`

Bei der Eingabe des regulären Ausdrucks muss man natürlich Zeichen, wie `>,<,&` maskieren, was den Ausdruck im Gegensatz zur herkömmlichen Nicht-XML Programmierung noch ein bisschen komplizierter macht.

Der Audruck, der bspw. auf alle XML Tags matched ist folgender: `<[>!]>` Hier sind, die für XML reservierten, Zeichen bereits maskiert.

XSLT Analyze String

Es gibt aber auch ein eigenes XSLT Konstrukt, unabhängig von XPATH, um reguläre Ausdrücke anwenden zu können. Der Quelltext dazu sieht so aus:

```
<xsl:template match="text()[parent::xml-code]">
  <xsl:analyze-string select="." regex="&lt;[^&gt;^!]+&gt;">
    <xsl:matching-substring>
      <b><xsl:value-of select="."/></b>
    </xsl:matching-substring>
    <xsl:non-matching-substring>
      <xsl:value-of select="."/>
    </xsl:non-matching-substring>
</xsl:template>
```

Hier wird jeder Textknoten eines Elements untersucht und falls ein XML-Tag enthalten ist, so wird dieses über die `xsl:matching-substring` Anweisung in ein `<b>` Tag gewrapped. Im nächsten Transformationsschritt wird dieses wiederum in Fettschrift dargestellt.

Das ist ein einfacher Syntax-Highlighter für XML Quelltexte. Auf diese Weise ist der Syntax-Highlighter für dieses PDF realisiert.

▶ Man kann auch die `xsl:analyze-string` Elemente in den `xsl:matching-substring` und `xsl:non-matching-substring` Elementen verschachteln, was natürlich noch wesentlich kompliziertere Problemstellungen erlaubt.

3.1.5 Modus vs. Tunnel Lösung

Als Entwickler wird man oft mit der Situation konfrontiert, dass es zur Lösung eines bestimmten Problems mehrere Möglichkeiten gibt und man eine davon auswählen muss.

Hier gilt es einen Trade-Off aus teils konkurrierenden Zielen zu finden, wie das folgende Beispiel zeigt. Betrachten wir ein Konto:

```
<konto nr="123">
    <inhaber>alex</inhaber>
    <vorgang>456</vorgang>
    <eintrag art="soll" datum="2019-05-06">20.56</eintrag>
    <eintrag art="soll" datum="2019-02-21">4.73</eintrag>
    <eintrag art="haben" datum="2019-01-14">1.68</eintrag>
    <eintrag art="soll" datum="2019-09-17">6.45</eintrag>
    <eintrag art="haben" datum="2019-01-03">12.38</eintrag>
    [...]
</konto>
```

Es gibt nun mehrere Möglichkeiten die Bilanz in zwei Dateien `soll.xml` und `haben.xml` aufzusplitten.

Schleife

```
<xsl:template match="/">
    <xsl:result-document href="soll.xml">
        <konto nr="{@nr}">
            <inhaber><xsl:value-of select="inhaber"/></inhaber>
            <vorgang><xsl:value-of select="vorgang"/></vorgang>
            <xsl:for-each select="konto/eintrag">
                <xsl:if test="@art='soll'">
                    <xsl:copy-of select="."/>
                </xsl:if>
            </xsl:for-each>
        </konto>
    </xsl:result-document>
    <xsl:result-document href="haben.xml">
        [...]
</xsl:template>
```

Hier werden die gemeinsamen Felder für das Konto `<inhaber>` und `<vorgang>` hartcodiert, was in dem einfachen Beispiel okay wäre. Aber eigentlich wollen wir ja Pull-Stylesheets vermeiden, vgl. Push vs. Pull Stylesheets on page 19.

Tunnel Parameter

```
<xsl:template match="/">
    <xsl:result-document href="soll.xml">
        <xsl:apply-templates>
```

```
            <xsl:with-param name="is-soll" select="true()" tunnel="yes"/>
        </xsl:apply-templates>
    </xsl:result-document>
    <xsl:result-document href="haben.xml">
        <xsl:apply-templates>
            <xsl:with-param name="is-soll" select="false()" tunnel="yes"/>
        </xsl:apply-templates>
    </xsl:result-document>
</xsl:template>

<xsl:template match="node()|@*">
    <xsl:copy>
        <xsl:apply-templates select="node()|@*"/>
    </xsl:copy>
</xsl:template>

<xsl:template match="eintrag">
    <xsl:param name="is-soll" tunnel="yes"/>
    <xsl:choose>
        <xsl:when test="$is-soll and @art='soll'">
            <xsl:copy-of select="."/>
        </xsl:when>
        <xsl:when test="not($is-soll) and @art='haben'">
            <xsl:copy-of select="."/>
        </xsl:when>
        <xsl:otherwise/>
    </xsl:choose>
</xsl:template>
```

Hier werden über eine Default-Kopierregel alle Knoten des XML Gerüsts kopiert, d.h. das Gerüst kann beliebig gross sein, und wir brauchen uns darum nicht kümmern. Insofern ist diese Lösung einen Schritt weit generischer, als die zuvor gezeigte.

Wir sind sogar sehr flexibel was Änderungen angeht, da im Eintrag-Template leicht weitere Logik implementiert werden kann, die über alle Zweige der bedingten Anweisung greift, bspw. mittels einer lokalen Variablen, die einen Berechnungsausdruck enthält.

Mode Attribut

(Noch) kürzer geht es dagegen mit dem *mode* Attribut:

```
<xsl:template match="/">
    <xsl:result-document href="soll.xml">
        <xsl:apply-templates mode="soll"/>
    </xsl:result-document>
    <xsl:result-document href="haben.xml">
        <xsl:apply-templates mode="haben"/>
    </xsl:result-document>
</xsl:template>

<xsl:template match="node()|@*" mode="#all">
    <xsl:copy>
        <xsl:apply-templates select="node()|@*" mode="#current"/>
    </xsl:copy>
</xsl:template>

<xsl:template match="eintrag[@art='soll']" mode="haben"/>
<xsl:template match="eintrag[@art='haben']" mode="soll"/>
```

Der Trick ist hier, dass im jeweils anderen Modus, gerade die ausgewählten Knoten nicht kopiert werden, was schliesslich die Eingabe korrekt in Soll und Haben aufteilt. Wenn wir weitere Logik hinzufügen wollen, müssten wir die Templates ein bisschen umbauen, was nicht weiter tragisch sein sollte.

Der Trade-Off besteht also aus den meist konkurrierenden Zielen Redundanz einzusparen, d.h. den Quelltext so kurz und knapp wie möglich zu gestalten, und einer zukünftigen leichten Erweiterbarkeit.

3.1.6 Identifikation mit `generate-id()`

Die `generate-id()` Funktion gibt es seit XSLT 1.0. Mit ihr kann eine Prüfsumme eines Knotens im Baum generiert werden.

Das funktioniert natürlich nur, wenn man bei der Auswertung dieses Wertes nicht den Kontext wechselt. D.h. z.B. dass ein Knoten in einem Baum, der in einer Variablen gespeichert ist, eine andere Prüfsumme bekommt, als derselbe Knoten im Kontext-Baum.

Beispiel Stückliste

Ein Anwendungszenario wäre bspw. die Generierung einer Target-ID für ein Bauteil in einer Stückliste. Das Bauteil ist nur einmal im System erfasst, hat also eine eindeutige ID, soll aber an mehreren Stellen in die Ausgabe (Eine Dokumentation für eine Maschine) generiert werden.

Die Id an einem Element `<part id="1234">` würde somit mehrfach in die XML Eingabe für einen XSL-FO Prozessor erscheinen und ist für Referenzen unbrauchbar geworden. Deshalb ist es ratsam beim Rendern der Bauteile eine neue Id zu vergeben, das kann z.B. mit den folgenden Templates (vereinfacht) passieren:

```
<xsl:key name="parts" match="part" use="@id"/>

<xsl:template match="part" mode="content">
  <!-- Ausgabe des Bauteils im Content Bereich -->
  <fo:block id="{generate-id()}">
    <fo:external-graphic xsl:use-attribute-sets="part.img"/>
  </fo:block>
</xsl:template>

<xsl:template match="part" mode="part-list">
  <!-- Ausgabe einer Liste mit allen Verweisen an unterschiedicher Stelle -->
  <fo:block>
    <xsl:for-each select="key('parts',@id)">
      <fo:page-number-citation ref-id="{generate-id()}"/>
    </xsl:for-each>
  </fo:block>
</xsl:template>
```

Beispiel Mantel Dokument

Im Bereich EDI Datenaustausch werden große XML Dateien versendet, die man auf einzelne Transmissions aufsplitten will, um sie in einer XML Datenbank abspeichern zu können. Die Struktur einer Datenübertragung könnte folgendermassen aussehen:

```
WRAPPER1
  SEQUENZ1
  SEQUENZ2
  SEQUENZ3
  WRAPPER2
    SEQUENZ1
    SEQUENZ2
    SEQUENZ3
    SEQUENZ4
    WRAPPER3
      SEQUENZ1
      SEQUENZ2
      CONTENT
        DATA1
        DATA2
```

```
        DATA3
        DATA4
        DATA5
      CONTENT
        DATA1
        DATA2
        DATA3
        DATA4
        DATA5
      WRAPPER4
        SEQUENZ1
      CONTENT
        DATA1
        DATA2
        DATA3
        DATA4
        DATA5
      [...]
```

Jedes einzelne *CONTENT* Element soll nun einen Mantel erhalten und separat in einer Datei abgelegt werden. Der "Umschlag" soll dabei alle Elemente des Rahmens der Transmission erhalten. Das ist alles auf der Descendant-Achse bis zum Element *WRAPPER3* , ausserdem noch die Elemente *SEQUENZ1* und *SEQUENZ2* , sowie das Element *WRAPPER4* mit Kind *SEQUENZ1* . Ohne groß auf die Performanz zu achten, könnte das recht einfach so realisiert werden:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform" version="2.0">

  <xsl:output method="xml" indent="yes"/>
  <xsl:strip-space elements="*"/>

  <xsl:template match="/">
    <xsl:apply-templates select="/WRAPPER1/WRAPPER2/WRAPPER3/CONTENT" mode="umschlag"/>
  </xsl:template>

  <xsl:template match="CONTENT" mode="umschlag">
    <xsl:result-document href="{concat(@id,'.xml')}">
      <umschlag>
        <metadaten><!-- einige Metadaten --></env:metadata>
        <nutzdaten>
            <xsl:apply-templates select="ancestor::WRAPPER1">
              <xsl:with-param name="this-id" select="generate-id()" tunnel="yes"/>
            </xsl:apply-templates>
        </nutzdaten>
      </umschlag>
    </xsl:result-document>
  </xsl:template>

  <xsl:template match="node()|@*">
    <xsl:copy>
      <xsl:apply-templates select="node()|@*"/>
    </xsl:copy>
  </xsl:template>

  <xsl:template match="CONTENT">
    <xsl:param name="this-element" tunnel="yes"/>
    <xsl:if test="$this-id = generate-id()">
      <xsl:copy>
        <xsl:apply-templates select="node()|@*"/>
      </xsl:copy>
    </xsl:if>
  </xsl:template>
```

```
</xsl:stylesheet>
```

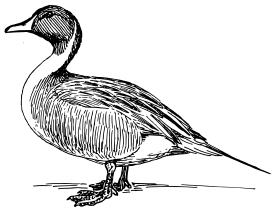
Im rekursiven Abstieg wird im Modus "umschlag" jedes *CONTENT* Element selektiert und in einen Umschlag verpackt. Der eigentlich Inhalt des Umschlags wird generiert, indem der gesamte XML Baum über die Standard-Kopierregel in das Element *<nutzdaten>* gesetzt wird. Dabei wird aber nur derjenige *CONTENT* Abschnitt evaluiert, der - zu der als Parameter übergebenen - generierten Id passt.

Verlinkung auf nächstes Verweisziel

Beispiel Publishing: In einer Publikation sind Grafiken vorhanden, auf die verlinkt werden soll. Das hört sich einfach an, jedoch gibt es auch Fälle, in denen ein solches Verweisziel mehrfach im Buch vorhanden ist.

Man sollte dann z.B. nicht auf dieselbe Grafik in Kapitel 1 verweisen, sondern besser auf die Grafik, die am nächsten liegt. Das kann in Blätterrichtung, aber auch entgegen der Blätterrichtung sein.

Abgesehen davon, dass es natürlich mehr Sinn macht, auf die Vorgänger-Grafik zu verweisen, weil der Leser ja beim Blättern diese schon begutachtet hat und nur zurückblättern muss (anstatt schnell vorzublättern und Inhalte zu überspringen), ist dieses Problem nicht ganz trivial:



1. Das nächst gelegene Verweisziel findet man über die *preceding* und *following* XPath-Achse:

```
<xsl:variable name="nearest-preceding"
              select="preceding::*[@id = current()][1]"/>
<xsl:variable name="nearest-following"
              select="following::*[@id = current()][1]"/>
```

2. Welches Verweisziel ist aber nun näher? Um diese Frage zu beantworten, müsste man irgendwie die Distanz zwischen den Knoten messen können.
3. Wenn man öfters mit der *preceding* und *following* Achse zu tun hat, merkt man schnell, dass die häufige Selektion in diesen beide Achsen auf die Performanz geht. Wie kann man die Performanz hier optimieren?

Neben der *generate-id()* Funktion werden zwei weitere - sehr clevere - XSLT Konstrukte zur Beantwortung dieser Fragen verwendet:

- *xsl:key* um die gesuchten Elemente zum schnellen Auffinden in einen Index aufzunehmen.
- Den relativ neuen *<<* -Operator, um die Postion des aktuellen Elements im DOM Baum zu bestimmen.

Mit diesen beiden Konstrukten, kann man beispielsweise die Anzahl aller Paras bis zu einer gewissen Position im DOM berechnen. Das geht so:

```
<xsl:key name="paras" match="p" use="true()"/>
[...]
<xsl:variable name="current-para-count"
              select="count(key('paras',true())[. &lt;&lt; current()])"/>
```

Zieht man den Para-Count das erste **Vorgänger-Verweisziels** davon ab, hat man einen Distanzwert bestimmt.

Jetzt kann man dasselbe für das erste **Nachfolger-Verweisziel** machen und vergleichen.

Mit diesem Vorgehen lässt sich schon Punk 2.) aus obiger Liste beantworten, denn das nächste Verweisziel, egal ob in Blätterrichtung oder entgegen der Blätterrichtung kann bestimmt werden.

Packt man diese Erkenntnisse in ein XSLT Stylesheet, dann sieht das ungefähr so aus:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    exclude-result-prefixes="xs"
    version="2.0">

    <xsl:key name="ids" match="@id" use="."/>
    <xsl:key name="paras" match="para" use="true()"/>

   <!-- Vergabe einer neuen, eindeutigen ID an den Verweiszielen, so dass auf das -->
   <!-- nächstgelegene Verweisziel verwiesen werden kann. -->

   <xsl:template match="@id[count(key('ids',.)) gt 1]">
        <xsl:attribute name="id" select="concat(.,'_',generate-id(parent::*))"/>
    </xsl:template>

    <!-- Vergabe einer neuen Ziel-ID an den Links -->

    <xsl:template match="@target-id[count(key('ids',.)) gt 1]">
        <xsl:variable name="nearest-preceding" select="preceding::*[@y.id = current()]"/>
        <xsl:variable name="nearest-following" select="following::*[@y.id = current()]"/>
        <xsl:choose>
            <xsl:when test="$nearest-preceding and $nearest-following">
              <xsl:variable name="current-para-count"
                            select="count(key('paras',true())[. &lt;&lt; current()])"/>
              <xsl:variable name="nearest-preceding-para-count"
                            select="count(key('paras',true())
                                 [. &lt;&lt; $nearest-preceding])"/>
              <xsl:variable name="nearest-following-para-count"
                            select="count(key('paras',true())
                                 [. &lt;&lt; $nearest-following])"/>
              <xsl:variable name="distance-to-preceding"
                            select="$current-para-count - $nearest-preceding-para-count"/>
              <xsl:variable name="distance-to-following"
                            select="$nearest-following-para-count - $current-para-count"/>
              <xsl:attribute name="target-id"
                            select="if ($distance-to-preceding le $distance-to-following)
                                    then concat($nearest-preceding/@y.id,
                                                '_',generate-id($nearest-preceding))
                                    else concat($nearest-following/@y.id,
                                   '_',generate-id($nearest-following))"/>
            </xsl:when>
            [...]
        </xsl:choose>
    </xsl:template>
```

3.1.6.1 XPath-Achsenbereich selektieren

Mit den folgenden Ausdrücken selektiert man ausgehend vom aktuellen Kontextknoten alle Knoten, die auf den betreffenden XPath Achsen liegen:

- `preceding-sibling::*`
- `following-sibling::*`
- `ancestor::*`
- `descendant::*`
- `ancestor-or-self::*`

- `descendant-or-self::*`

Schwieriger wird es, wenn man nur einen bestimmten Bereich auswählen will und nicht alle Knoten bis zum Anfang bzw. Ende der Achse.

Die Selektion eines Bereichs kann man z.B. beim Strukturieren einer Sequenz aus Überschriften und Paras gebrauchen:

```
<body>
  <h1>Überschrift 1</h1>
  <p>Para 1.1</p>
  <p>Para 1.2</p>
  <h1>Überschrift 2</h1>
  <p>Para 2.1</p>
  <p>Para 2.2</p>
  <h1>Überschrift 3</h1>
  <p>Para 3.1</p>
  <p>Para 3.2</p>
</body>
```

Die Aufgabe besteht nun darin, jeweils einen `section` -Abschnitt einzuführen, so:

```
<section>
  <h1>Überschrift 1</h1>
  <p>Para 1.1</p>
  <p>Para 1.2</p>
</section>
<section>
  <h1>Überschrift 2</h1>
  <p>Para 2.1</p>
  <p>Para 2.2</p>
</section>
<section>
  <h1>Überschrift 3</h1>
  <p>Para 3.1</p>
  <p>Para 3.2</p>
</section>
```

Ein XSLT Template das dieses Unterfangen bewerkstelligen könnte, sieht bspw. so aus:

```
<xsl:template match="body">
  <xsl:copy>
    <xsl:for-each select="h1">
      <section>
        <xsl:copy/>
        <xsl:apply-templates select="following-sibling::*[not(self::h1)]
                                     [preceding-sibling::h1[1] is current()]"/>
      </section>
    </xsl:for-each>
  </xsl:copy>
</xsl:template>
```

Hier werden im `apply-templates` Call alle Nachfolger-Knoten ausgehend vom aktuellen Kontextknoten selektiert, die keine Überschriften sind, und die als erste Vorgängerüberschrift den aktuellen Kontextknoten haben. Das ist genau der Bereich bis zur nächsten Überschrift.

Funktionen und Module

Funktionen

Um bestimmte Abschnitte des XQuery Programm wiederverwendbar zu machen, stehen Funktionsdeklarationen zur Verfügung. Eine einfache Funktion wäre z.B. diese hier:

```
declare function local:wrap-header($json) {
  xdmp:add-response-header("Pragma", "no-cache"),
  xdmp:add-response-header("Cache-Control", "no-cache"),
  xdmp:add-response-header("Expires", "0"),
  xdmp:set-response-content-type('text/json; charset=utf-8'),
  $json
};
```

Sie wickelt um einen JSON String eine passende Header Information.

Damit die Funktion eingebunden werden kann, muss ein passender Namespace deklariert werden:

```
declare namespace local = 'local:';
```

Nicht nur bzgl. Wiederverwendbarkeit sind Funktionen praktisch, sondern auch um ganz elementare Konstrukte, wie *`while...do`* Schleifen, zu realisieren.

Dazu nutzt man - wie in der funktionalen Programmierung üblich - die Rekursion:

```
declare function local:ist-letzter-wert-in-kette($glied) {
  let $wert := local:komplizierte-berechnung($glied),
    $naechstes-glied := local:komplizierte-berechnung-der-position($glied),
  return
    if ($naechstes-glied and not($wert = 'foobar')) then
      local:durchlaufe-kette($naechstes-glied)
    else
      $wert = 'foobar'
};
```

In diesem kleinen Schnippsel sind schon einige Besonderheiten von XQuery zu sehen. Variablenzuweisungen geschehen mit einem Doppelpunkt, Vergleiche dagegen nur mit einem einfachen "=". Statements werden mit einem Komma getrennt.

Funktionsaufrufe im XPath

Wenn eine Funktion auf einer Kontenmenge operiert, dann kann der Funktionsaufruf auch an einen Pfadselektor gehangen werden, bspw. so:

```
<xsl:value-of select="sum($current/betrag[xs:decimal(.) gt 0]/xs:decimal(.))"/>
```

Hier werden die *`betrag`* -Knoten eines zuvor selektierten Teilbaums, der in der Variablen *`$current`* abgespeichert ist, aufsummiert - aber nur wenn der Wert größer als 0 ist.

Der Funktionsaufruf ist hier ein Type-Cast auf einen Dezimalwert, um eine gewisse Rechengenauigkeit zu gewährleisten.

Die Filterung auf positive Werte ist dabei noch gewöhnlich formuliert:

```
[xs:decimal(.) gt 0]
```

`xs:decimal` nimmt den aktuell ausgwählten Knoten und macht einen Dezimalwert daraus, um ihn mit 0 zu vergleichen.

Falls hier an den Typkonstruktor `xs:decimal` ein nicht-unterstütztes Format übergeben wird, bspw. ein String, dann wird ein **fatalen Fehler geworfen und das Programm bricht ab**.

Der Funktionsaufruf kann aber auch als Pfadselektion an einem XPath angebracht werden:

```
/xs:decimal(.)
```

Im Fehlerfall wird **der fehlerhafte Wert nicht summiert und das Programm läuft weiter**.

```
$current/betrag[xs:decimal(.) gt 0]
```

Auf herkömmlichen Weg würde man eine Schleife verwenden, die alle Werte auf deren Dezimalwert abbildet:

```
sum(for $xin $current/betrag[xs:decimal(.) gt 0]
return xs:decimal($x))
```

Das ist ein bisschen komplizierter, gewährleistet aber eine bessere Robustheit der Programmierung.

Funktionsaufrufe als Pfadselektoren brechen bei einem Fehler in der Funktion - ohne explizite Ausnahmebehandlung - **nicht** ab.

- ▶ Falls mehr Robustheit gefordert ist, sollte man über Ergbnisknotenmengen iterieren und Funktionsaufrufe auf herkömmlichem Weg absetzen.

Betrachten wir folgendes XQuery-Schnippsel:

```
collection("/abrechnung")[vorgangsnummer[.=(3, 8,9,10)]/xdmp:node-collections(.)
                          [starts-with(., '/buchung')]/xdmp:collection-delete(.)
```

Hier sind alle Abrechnungen in einer Collection `/abrechnung` gespeichert. Die Abrechungen mit den Vorgangsnummern `3,8,9` und `10` sollen herausgefischt werden. Diese Abrechungen können auch in verschiedenen Collections verwaltet werden, bspw. mittels eines Collection-Typs "Buchung". Eine Buchung-Collection sammelt alle Abrechungen, die an einem bestimmten Buchungstag getätigt wurden. Wir gehen jetzt davon aus, dass alle Abrechungen `3,8,9,10` an einem bestimmten Buchungstag getätigt wurden - und nur diese. Aus irgendeinem Grund wollen wir diese Buchung nun löschen. Das macht genau der obige Einzeiler. Der Filter:

```
collection("/abrechnung")[vorgangsnummer[.=(3, 8,9,10)]
```

gibt eine Knotenmenge zurück. Das gefilterte Funktionsergebnis

```
xdmp:node-collections(.)[starts-with(., '/buchung')]
```

ist auch eine Knotenmenge. Normalerweise bräuchten wir also Schleifen, um über diese Mengen zu iterieren. Das würde irgendwie so aussehen

```
let
  $filtered-collection := collection("/abrechnung")[vorgangsnummer[.=(3, 8,9,10)],
  $collections-to-be-deleted :=
  distinct-values(
    for $xin $collection
       return (
         for $yin xdmp:document-get-collections(fn:document-uri($x))
                                                [starts-with(., '/buchung')]
           return
             $y
       )
    )
return (
      for $culpritin $collections-to-be-deleted
        return xdmp:collection-delete($culprit)
    )
```

Der Quelltext ist zwar so wesentlich länger, aber auch weniger geübte XPath-Experten erkennen leicht, um was es geht.

Module

Um eine XQuery Anwendung zu modularisieren, können einzelne Skripte in Module ausgelagert werden. Ein Modul, z.B. `common.xqy` , wird dabei über einen eigenen Namespace deklariert:

```
module namespace common = "https://www.tekturcms.de/common";
```

Dieses Modul kann dann in anderen Skripten eingebunden werden:

```
import module namespace common = "https://www.tekturcms.de/common" at "common.xqy";
```

Funktionen und Variablen werden dann mir dem Namespace geprefixt aufgerufen:

```
Funktionsaufruf: common:wrap-response-header(...)
```

```
Variablenauswertung: $common:collection-books
```

3.1.7 Webservice Calls mit `doc()` und `unparsed-text()`

Eine verbreitete Paxis ist es, mit der Funktion `document()` oder kurz `doc()` entfernte Ressourcen in die Transformation einzubinden. Bei einer Schematron-Validierung, würde bspw. eine Regel, wie:

```
<sch:not-assert id="personal-check"
    role="error"
    test="doc(concat('https://tekturcms.de/personal.xqy?personal-id=',personal-id))/kuendigung">
        Angestellter mit ID "<sch:value-of select="personal-id"/>" hat gekündigt!
</sch:not-assert>
```

einen entferneten Webservice aufrufen und prüfen, ob für den Angestellten mit `personal-id` eine Kündigung vorliegt. Ist dies der Fall, so ist die negative Zusicherung `not-assert` nicht erfüllt, und die Schematron Regel feuert - was sich wohl im einfachsten Fall in einem Logfile Eintrag äussern sollte.

Was vermutlich viele noch nicht kennen - ich nehme jetzt einfach mal an, dass mein bisheriger Kenntnisstand dem der Mehrheit der XML-Entwickler entspricht - ist der Umstand, dass auch die Funktion `unparsed-text()` eine URL als Parameter nimmt:

```
<xsl:template match="angestellter"
    <xsl:copy>
        <xsl:apply-templates select="node()|@*"/>
        <hat-gekuendigt>
            <xsl:sequence select="json-to-xml(
                                  unparsed-text(
                                  concat('https://tekturcms.de/personal.xqy?personal-id=',
                                  personal-id))))/descendant::*[@key='gekuendigt']/text()"/>
        </hat-gekuendigt>
    </xsl:copy>
</xsl:template>
```

Während mit `doc()` oder `document()` ein zurückgeliefertes XML Fragment prozessiert wird, erwartet `unparsed-text()` z.B. einen **JSON-String**, der dann mittels der Funktion `json-to-xml()` nach XML konvertiert werden kann.

Beispielsweise könnte die Gegenseite zum `angestellter` Template mittels XQuery folgendermassen realisiert sein:

```
xquery version "1.0-ml";

declare variable $personal-id := xdmp:get-request-field('personal-id');

let $gekuendigt := if (collection('/personal')/*[personal-id = $personal-id and
                                                 fn:exists(kuendingung)] then
                                                 'ja' else 'nein'
return
    common:render-response(concat('{"gekuendigt":"',$gekuendigt,'",
                                    "personal-id":"',$personal-id,'"}'))
```

(Mehr zu XQuery und den hier verwendeten Konstrukten, wie `render-response()`)

Das zurückgeklieferte JSON Dokument sieht dann so aus:

```
{"gekuendig":"ja","personal-id":"q5687500"}
```

Konvertiert nach XML erhält man eine Map Struktur:

```
<map xmlns="http://www.w3.org/2005/xpath-functions">
    <string key="gekuedigt">ja</string>
    <string key="personal-id">q5687500</string>
</map>
```

was den Selektorausdruck im obigen XPATH erklärt:

```
json-to-xml(
unparsed-text(
concat('https://tekturcms.de/personal.xqy?personal-id=',
personal-id))))/descendant::*[@key='gekuendigt']/text()
```

Resultat der Konvertierung wäre - wie erwartet - ein um das `gekuendigt` Flag erweitertes `<angestellter>` Element:

```
<angestellter>
    <perosnal-id>q5687500</perosnal-id>
    <name>Alex</name>
    [...]
    <gekuendigt>nein</gekuendigt>
</angestellter>
```

Sicherlich wird der XML Entwickler eine XML Datenbank, wie MarkLogic, vorziehen und sich gleich XML Fragmente ausliefern lassen. Tektur ist aber bspw. mit MongoDB[73] realisiert, die auf JSON arbeitet... Nicht zuletzt deshalb finde ich JSON Verarbeitung mit XSLT recht spannend.

3.1.8 Stylesheet-Parameter auf der Kommandozeile

Beim XSLT-Call auf die Einsprungdatei (Hauptstylesheet) könnnen auf der Konsole Parameter mitgegeben werden. Diese werden normalerweise im Hauptstylesheet deklariert, können aber auch in Sub-Stylesheets untergebracht werden, wenn diese nur lokal verfügbar sein sollen.

Ein Parameter kann z.B. so aussehen:

```
<xsl:param name="test-param" required="no" select="'Hallo'"/>
```

Wenn das Feld `@required` den Wert `yes` zugewiesen bekommt, dann darf kein Default-Wert im `@select` Attribut angegeben werden. (Der Default könnte natürlich auch als PCDATA Inhalt innerhalb der Tags stehen.)

Ein Saxon-Aufruf auf der Kommandozeile, der diesen Parameter bedient, sieht z.B. so aus:

73) https://www.mongodb.com/

```
saxon -it:main -o:out.xml testparam="Servus!"
```

XPath als Parameterwert

Jetzt gib es hier aber auch noch einige versteckte Funktionalitäten, bspw. möchte man einen XPath-Ausdruck an das Stylesheet übergeben, der vom Hauptprogramm vllt. unter Zuhilfenahme einer Datenbankabfrage berechnet wird.

Im einfachsten Fall ist so ein XPath ein boolscher Wert `true()` Dieser wird an das Stylesheet korrekt übergeben, indem der XPath Kennzeichner `?` vorangestellt wird, so:

```
saxon -it:main -o:out.xml ?testparam=true()
```

Wegen des `?` -Prefix versteht Saxon sofort, dass es sich um einen XPath handelt. Ein `xsl:choose` Statement wie das folgende würde den korrekten Fall liefern:

```
[...]
<xsl:param name="test-param" required="yes"/>
[...]
<xsl:choose>
    <xsl:when test="$test-param">Gut</xsl:when>
    <xsl:otheriwse>Schlecht</xsl:otheriwse>
</xsl:choose>
[...]
```

Würde man das Prefix nicht setzen und bspw. dieses `Name=Wert` Paar senden:

```
saxon -it:main -o:out.xml testparam=false
```

So würde auch der gute Zweig ausgeführt, da der Nicht-Leerstring als wahr interpretiert werden würde, was aber falsch ist.

Typsicher kann man natürlich auch programmieren und an den Parameter noch eine Typendeklaration heften, was die Fehlersuche erleichtert:

```
<xsl:param name="test-param" required="yes" as="xs:boolean"/>
```

Um noch ein komplexeres Szenario für einen übergebenen XPath zu zeigen, betrachten wir die folgende Parameterübergabe:

```
saxon -it:main -o:out.xml ?testparam="//descendant::buch[id=$id-aus-webservice-abfrage]"
```

Die Shell Variable `$id-aus-webservice-abfrage` wird bei der Ausfühung des Shell-Skripts mit dem Ergebniswert eines vorangegangenen Webservice-Calls bestückt und an das XSLT Stylesheet übergeben. Dieses arbeitet auf einer Bücher-Liste und formatiert das ausgewählte Buch z.B. als HTML Seite:

```
<xsl:param name="test-param" required="yes" />

<xsl:template name="main">
    <xsl:apply-templates select="$test-param"/>
```

```
</xsl:template>
```

Clark Notation

Bei der Clark Notation[74] kann dem Elementbezeichner mittels geschweifter Klammerung der benötigte Namespace vorangestellt werden. Das sieht z.B. wie folgt aus - wenn man auch gleich noch das `?` -Prefix voranstellt:

```
?{http://www.tekturcms.de/namespaces/mein-namespace}spezieller-verarbeitungs-modus=true()
```

Im aufgerufenen Stylesheet wird zunächst der Namespace deklariert, dann der Parameter gesetzt und schliesslich die Logik angegeben:

```
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    xmlns:tk="http://www.tekturcms.de/namespaces/mein-namespace"
    exclude-result-prefixes="xs"
    version="2.0">

    <xsl:param name="tk:spezieller-verarbeitungs-modus" select="false()" as="xs:boolean"/>

    <xsl:template name="main">
        <xsl:choose>
            <xsl:when test="$tk:spezieller-verarbeitungs-modus">Gut</xsl:when>
            <xsl:otheriwse>Schlecht</xsl:otheriwse>
        </xsl:choose>
    </xsl:template>

</xsl:stylesheet>
```

Diesen Sonderfall braucht man eigentlich nur, wenn man in eine bestehende Transformation einen Parameter einführen will, dessen Bezeichner schon existiert, um so einen Namenskonflikt zu vermeiden - gesehen bspw. bei komplexen Schematron Regeln.

3.1.9 Leerzeichenbehandlung

Die Leerzeichenbehandlung ist im Bereich Publishing ein heisses Thema. Einerseits sollte der Redakteur dafür sorgen, dass keine überflüssigen Leerzeichen in die Bedatung gelangen, andererseits werden diese aber durch Editoren und Transformationsstrecken hinzugemogelt.

Diese unerwünschten Leerzeichen will man bei der Publikation wieder loswerden. Hierzu stellt XSLT verschiedene Mittel bereit:

1. Mit `xsl:strip-space` und `xsl:preserve-space` kann in der Stylesheet-Deklaration festgelegt werden, in welchen Elementen Leerzeichen-artige Zeichen, das sind z.B. Spaces und Zeilenumbrüche, ggf. auf ein Leerzeichen normalisiert werden sollen. Andersrum kann man auch angeben, wo das explizit nicht geschehen soll.
2. Die Funktion `fn:normalize-space` ermöglicht diese Funktionalität in XPath Abfragen und mit `fn:translate` können Leerzeichen auf andere Zeichen, wie z.B. auch auf den Leerstring, abgebildet werden.
3. Will man nur die Leerzeichen am Anfang oder am Ende eines PCDATA Elements loswerden, empfiehlt sich ggf. auch ein Blick in die FunctX Bibliothek von Priscillia Wamsley zum Thema Trimming[75].

74) http://www.jclark.com/xml/xmlns.htm
75) http://www.xsltfunctions.com/xsl/functx_trim.html

Leerzeichen am Satzanfang

Schnell stellt man fest, dass diese Hausmittel nicht ausreichen, wenn man die Leerzeichen am Anfang einer verschachtelten Inline-Struktur loswerden will, wie z.B. dieser hier:

```
<title>•<i>••<!-- comment --><cap>•A</cap>lfons Bliemetsrieders </i>Tagebuch</title>
```

Die Leerzeichen, markiert als schwarze Punkte •, haben sich eingeschlichen. Man kann hier nicht obige Funktionen pauschal anwenden, denn z.B. das Leerzeichen hinter `Bliementsrieder` ist durch ein Editierproblem entstanden. Hier wurde versehentlich das Leerzeichen kursiv mitformatiert. Würde man auf dem `<i>` Tag normalisieren, dann würde dieses Leerzeichen verschluckt werden.

Zweistufige Leerzeichen-Eliminierung

Mein Ehrgeiz für dieses Problem wurde durch die sehr gute Lösung meines Kollegen geweckt, der einen anderen Programmierstil pflegt. Deshalb musste ich beweisen, dass man auch "old-school" regelbasiert mit XSLT zu einer vernünftigen Lösung kommt. Glücklicherweise gibt es mittlerweile sehr ausgeklügelte XSLT Konstrukte.

Wenn man sich das obige Beispiel anschaut, dann lässt sich die Aufgabe in zwei Teile zerlegen:

1. Entferne alle Textknoten unterhalb von `<title>` bis zum ersten Textknoten, der auch Buchstaben und sichtbare Zeichen enthält.
2. Danach kannst Du am ersten Textknoten unterhalb von `<title>` die führenden Leereichen abschneiden.

Hört sich simpel an, ist es aber leider nicht.

Zunächst recherchierte ich, ob denn auch wirklich an einem PCDATA Element nur ein Textknoten dranhängt. Diese Information war nötig, weil mein erster Algorithmus noch nicht ganz so ausgefeilt war, wie in den zwei Punkten oben beschrieben.

Man kann in einer Transformation mehrere Textknoten hintereinander erzeugen, wie:

```
<xsl:value-of select="'erster Textknoten'"/><xsl:value-of select="'zweiter Textknoten'"/>
```

Diese werden aber bei einem *"Save-Load Cycle"* zu einem Textknoten normalisiert. So steht das zumindest in der DOM Core Spezifikation[76]. Inwieweit das dann in den XML Prozessoren umgesetzt ist, musste noch geprüft werden. Dazu habe ich den Saxon Quellcode herangezogen:

76) https://www.w3.org/TR/DOM-Level-3-Core/core.html#Document3-normalizeDocument

Quellcode Schnippsel aus dem Saxon XSLT Prozessor, das zeigt, dass der EndElement-Listener im Parser einen Normalisierungsschritt auf den beteiligten DOM Knoten aufruft.

```
195    /**
196     * End of an element.
197     */
198
199    @Override
200    public void endElement() throws XPathException {
201        nsStack.pop();
202        if (canNormalize) {
203            try {
204                currentNode.normalize();
205            } catch (Throwable err) {
206                canNormalize = false;
207            }       // in case it's a Level 1 DOM
208        }
209
210        currentNode = currentNode.getParentNode();
211        level--;
212    }
213
```

figure 9: endElement() Funktion im Saxon XSLT Prozessor

Die Normalisierungsfunktion lässt Mike Kay dann mit einem aussagekräftigen Kommentar frei...

Methodenrumpf der Normalisierungsfunktion im Saxon XSLT Prozessor

```
160
161    /**
162     * Puts all <code>Text</code> nodes in the full depth of the sub-tree
163     * underneath this <code>Node</code>, including attribute nodes, into a
164     * "normal" form where only structure (e.g., elements, comments,
165     * processing instructions, CDATA sections, and entity references)
166     * separates <code>Text</code> nodes, i.e., there are neither adjacent
167     * <code>Text</code> nodes nor empty <code>Text</code> nodes.
168     *
169     * @since DOM Level 2
170     */
171
172    @Override
173    public void normalize() {
174        // null operation; nodes are always normalized
175    }
176
```

figure 10: normalize() Funktion im Saxon XSLT Prozessor

Damit war ich zufrieden - ob das jetzt stimmt oder nicht, ist glücklicherweise für die endgültige Lösung irrelevant.

Mein erster Versuch alle Textknoten auszuschneiden, die nur Leerzeichen enthalten, sah so aus:

```
<!-- Entferne alle Leerzeichen-Only Knoten, die kein Vorgänger Inline-Element haben -->
<xsl:template match="text()[ancestor::title
                            and not((.|..)/preceding-sibling::node()[1][self::*]) and
                                not(translate(.,' ',''))]"/>
```

- Die erste Bedingung prüft, ob sich der Textknoten irgendwo unterhalb von *<title>* befindet.
- Die zweite Bedingung prüft, ob als unmittelbarer Vorgänger ein Inline-Element existiert. Gesetzt den Fall, dass aneinander angrenzende Textknoten zu einem Textknoten zusammengefasst sind - wie oben recherchiert - würde das im Negativfall bedeuten, dass wir uns am Satzanfang befinden.
- Die dritte Bedingung prüft, ob es sich um einen Knoten handelt, der nur aus Leerzeichen besteht. Hier müssten streng genommen auch noch Zeilenumbrüche aufgelistet sein.

Leider konnte dadurch der folgende - Nicht-Real-Welt - Testfall nicht gelöst werden:

```
<title>•<b>•Fettes</b><b><i><b><i> </i></b></i>Editierproblem</b></title>
```

Das Leerzeichen im `<i>` Tag wurde verschluckt. Das kam wegen der zweiten Bedingung, die nur maximal eine Verschachtelungsebene beachtet. Man könnte zwar den Ausdruck noch aufbohren, z.B. so:

```
not((.|..|../..|../../..)/preceding-sibling::node()[1][self::*])
```

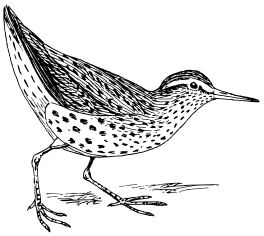
Das sieht aber schon wirklich sehr unschön aus.

Da ich mir aber zuvor schon den zweiten Schritt überlegt hatte, der so aussieht:

```
<!-- Entferne am ersten Textknoten unterhalb von title führende Leerzeichen -->
<xsl:template match="text()[current() is ancestor::title[1]/(descendant::text())[1]]"
              priority="10"  mode="pass-2">
    <xsl:value-of select="replace(.,'^\s+','')"/>
</xsl:template>
```

...fiel mir schliesslich die Korrektur für den ersten Schritt leicht:

```
<xsl:template match="text()[current() &lt;&lt; ancestor::title[1]/
                        (descendant::text()[normalize-space(.)])[1]]" mode="pass-1"/>
```



- Ein Test `text()[normalize-space(.)` genügt, um festzustellen, ob der Textknoten nicht nur Leerzeichen enthält.
- Andersrum prüft man mit `text()[not(translate(.,' ',''))` ob der Textknoten nur aus Leerzeichen besteht.
- Das Flachklopfen einer Sequenzmenge mittels `()` , wie in `(descendant::text())` ist notwendig, damit man auch wirklich nur das erste Element des Descendant-Lookups bekommt.
- Die `fn:current()` Funktion wird viel zu selten benutzt... damit erspart man sich eine Variablendeklaration im Rumpf der Regel.
- Den coolen `<<` Operator, der prüft, ob ein Knoten vor einem anderen kommt, muss man in einem Match-Statement escapen.

Abschliessend ist noch der ganze Quelltext der Lösung abgebildet. Dieser zeigt auch nochmal das Pattern bzgl. der Vortransformationen (imported) on page 50:

```
<?xml version="1.0" encoding="UTF-8"?>
<xsl:stylesheet xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
    xmlns:xs="http://www.w3.org/2001/XMLSchema"
    exclude-result-prefixes="xs"
    version="2.0">

    <xsl:template match="text()[current() &lt;&lt; ancestor::title[1]/
                            (descendant::text()[normalize-space(.)])[1]]"
                  mode="pass-1"/>

    <xsl:template match="text()[current() is ancestor::title[1]/(descendant::text())[1]]"
                  priority="10"  mode="pass-2">
        <xsl:value-of select="replace(.,'^\s+','')"/>
```

```
    </xsl:template>

    <xsl:template match="/">
        <xsl:variable name="pass-1">
            <xsl:apply-templates mode="pass-1"/>
        </xsl:variable>
        <xsl:apply-templates select="$pass-1" mode="pass-2"/>
    </xsl:template>

    <xsl:template match="node()|@*" mode="#all">
        <xsl:copy>
            <xsl:apply-templates select="node()|@*" mode="#current"/>
        </xsl:copy>
    </xsl:template>

</xsl:stylesheet>
```

Exkurs: Performanz der Descendant-Achse

Aus Verlegenheit hatte ich die komplexere Match-Bedingung des ersten Schritts gegen einen einfachen Pfadselektor ausgewechselt, also das....

```
match="text()[current() is ancestor::title[1]/(descendant::text())[1]]"
```

... gegen das ...

```
match="title/(descendant::text())[1]"
```

... ersetzt.

Damit lief die Transformation aber in ein exponentiellen Performanzproblem!

▶ In einer Match-Bedingung sollte niemals die Descendant-Achse im Pfadselektor auftauchen!

Ansonsten performt die Lösung auch bei mehreren Tausend Titeln in Sekundenbruchteilen.

Leerzeichen vor einem Inline-Element

Auch die Leerzeichen vor einem Inline-Element sind prüfenswert. Insbesondere um zu sehen, ob keines vergessen wurde. Eine naive Herangehensweise wäre bspw. ausgehend vom Inline-Element zu prüfen, ob ein Vorgänger Textknoten mit einem Leerzeichen abschliesst:

```
ends-with(preceding-sibling::text()[1],' ')
```

Dabei werden aber nur die Text-Vorgängerknoten in der Zeile betrachtet, etwaige Inline-Elemente werden aussen vor gelassen. D.h. der folgende Para würde nach diesem XPath Ausdruck noch als gültig erkannt, da nach dem "Hallo " ein Leerzeichen steht.

```
<p>Hallo <b>fetter Text</b><link>link text</link>
```

Offensichtlich fehlt aber ein Leerzeichen vor dem `<link>` -Element.

Um zu prüfen, ob der unmittelbare Vorgängerknoten ein Textknoten ist und ob dieser mit einem Leerzeichen abschliesst, kann man diesen XPath Ausdruck verwenden:

```
ends-with(preceding-sibling::node()[1][self::text()],' ')
```

Oder klarer:

```
preceding-sibling::node()[1][self::text()]/ends-with(.,' ')
```

Da wir aber bisher immer nur die preceding-sibling Achse betrachten, entgehen uns Leerzeichen, die in zuvor gesetzten Inline-Elementen vorkommen, bspw. hier:

```
<p>Hallo <b>fetter Text </b><link>link Text</link></p>
```

Deshalb müsste man eigentlich den XPath Ausdruck noch erweitern:

```
preceding-sibling::node()[1][self::* or self::text()]/ends-with(string(.),' ')
```

Dabei ist zu beachten, dass der string()-Cast auch noch verschachtelte Inline Strukturen flachklopfen würde.

Jetzt könnte man sich denken, dass man ja eigentlich diesen Ausruck verkürzen könnte zu:

```
preceding-sibling::node()[1]/ends-with(string(.),' ')
```

Dann würde aber auch die folgende Zeile erfolgreich geprüft:

```
<p>Hallo <b> fetter Text <i>Hallo</i></b><!-- Hallo --><link>link Text</link></p>
```

Denn der Kommentar schliesst mit einem Leerzeichen ab. Auch Kommentare und Processing Instructions sind vom Typ `node()` .

3.1.10 Mit `translate` Zeichen ersetzen

Die `fn:translate()` XPath Funktion gibt es schon seit XSLT1.0. Mit den moderneren Varianten, wie `fn:replace()` die auch ERROR! Linktarget does not exist verstehen, gibt es aber für die meisten Anwendungsfälle mächtigere Alternativen.

Jedoch kann man mit `translate` mehrere Zeichen auf einmal ersetzen. Mit `replace` müsste man dagegen die Funktionen verschachteln, so wie in diesem Beispiel:

```
<xsl:function name="dy:make-url">
        <xsl:param name="root"/>
        <xsl:value-of select="replace(replace(replace(replace($url-pattern,
                                '%id%', $root/@id),
                                '%version%',$root/@version),
                                '%language%',$root/@language),
                                '%variant%',$root/@variant)"/>
    </xsl:function>
```

Priscilla Walmsley bietet aber auch eine Funktion, die mehrere Replacements vornehmen kann[77].

Das Ersetzen von Einzel-Zeichen macht man am besten mit `fn:translate()` .

In der Indexsortierung des Buchindex wurden in XSLT1.0 Stylesheets die Gruppentitel der Indexrubriken mittels dieser Funktion normalisiert. Um bspw. die Zeichen `AÄÅaäå` (beachte das schwedische Bolle-A[78] auf den Gruppentitel `A` abzubilden, würde folgender `translate` -Aufruf genügen:

```
<xsl:value-of select="translate(.,'AÄÅaäå','A')/>
```

Das große `A` wäre nun der Gruppentitel für unsere Zeichen. Natürlich will man nicht jedes Zeichen einzeln mappen, sondern eine globale Regel für das ganze Alphabet angeben. Das könnte bspw. für die französischen Kleinbuchstaben so aussehen:

```
<xsl:variable name="kleinbuchstaben"
              select="'abcdefghijklmnopqrstuvwxyzÀàÆæÇçÉéÈèÊêËëÎîÏïÖöÔôŒœÙùÜüÛûŸÿ'/>
<xsl:variable name="gruppentitel"
              select="'ABCDEFGHIJKLMNOPQRSTUVWXYZAAAACCEEEEEEEEIIIIOOOOOOUUUUUUYY'/>
<xsl:value-of select="translate(.,$kleinbuchstaben,$gruppentitel)"/>
```

Hier kommt es auf die Reihenfolge der Zeichenketten an. Ausserdem müssen die Strings gleich lang sein.

- Solche Zeichenketten schreibt man am besten in eine sprachabhängige separate XML Datei, die auch ausserhalb des Quelltexts gepflegt werden könnte.

Auch die Prüfung, ob ein Wort mittels Leerzeichen im Satz abgesetzt ist, lässt sich mittels `fn:translate` realisieren. Hier wird sicherlich auch eine Prüfung notwendig

77) http://www.xsltfunctions.com/xsl/functx_replace-multi.html
78) https://de.wikipedia.org/wiki/%C3%85

sein, die prüft, ob eine Zeichenkette schon mit einem Leerzeichen (unter Ausschluss von Interpunktionszeichen) beginnt:

```
test="not(starts-with(replace(
                     replace(.,$common-punctuation-marks,''),$whitespaces,' '),' '))"
```

mit dieser Variablenbelegung:

```
<sch:let name="common-punctuation-marks"
         value="'.,!?()[]{}&#xFF02;&#x60;&#x22;:;##©—–-+…=«»¿&lt;&gt;&quot;%‰®™|/'"/>
<sch:let name="whitespaces"
         value="'&#32;         '"/>
```

Dem Einsatzbereich von `fn:translate()` ist mit genügend Lösungsphantasie keine Grenzen gesetzt.

3.1.10.1 Spass mit dem Sequenzvergleich

Weil man immer wieder mal mit **XPath** oder **XQuery** beim Sequenzvergleich auf die Nase fliegt, hier die wichtigsten Kniffe in Form einer XQuery Sitzung (gilt auch für XPath):

```
xquery version "3.1";

(: Prüfen, ob ein Elementin einer Sequenz vorkommt :)
('a','b') = 'a',          (: => true()  :)

(: Beachte, aber auch... :)
('a','a') = 'a',          (: => true()  :)
not(('a','b') != 'a'),    (: => false() :)

(: Prüfen, ob ein Element mit jedem Element einer Sequenz matcht :)
not(('a','a') != 'a'),     (: => true()  :)

(: Prüfen, ob Elemente einer Sequenzin einer anderen enthalten sind :)
('a','b','c') = ('a','b','d','e','f'),  (: => true() :)

(: Prüfen, ob zwei Sequenzen gleich sind :)
deep-equal(('a','b','c'), ('a','b','c')),  (: => true()  :)

(: Prüfen, ob zwei Sequenzen die gleichen Elemente enthalten :)
('a','b','c') = ('a','b','c') and
count(distinct-values(('a','b','c'))) =
count(distinct-values(('a','b','c')))      (: => true() :)
```

Sollte man nicht nur atomare Werte vergleichen wollen, so ist man sicherlich auch mit der FunctX[79)] Bibliothek gut bedient.

3.1.11 Character Mappings in der Ausgabe

Die Umstellung von XHTML auf HTML5 war nicht unbedingt ein Fortschritt, wenn es um die Verarbeitung der Strukturen mittels XSLT geht. Denn ein HTML5 Dokument kann im Gegensatz zu XHTML auch nicht abgeschlossene Tags beinhalten.

79) http://www.xsltfunctions.com/xsl/c0015.html#c0052

Man muss also zunächst das HTML5 mittels z.B. HTML Tidy[80] valide machen und nach XHTML überführen, bevor man es dann per XSLT weiterverarbeiten kann. Natürlich frägt sich jetzt der geneigte Leser, warum man das will - schließlich handelt es sich bei HTML5 ja schon um ein Ausgabeformat.

Mir fallen auf Anhieb zwei Use Cases ein:

- Reverse Engineering eines Ausgabepakets der Konkurrenz, um die Daten in das eigene CCMS zu importieren
- Web scraping mittels XSLT

Aber auch schon einen Schritt vorher, wenn Saxon den Ausgabebaum für HTML5 schreibt, können Probleme auftreten. Insbesondere wenn sich unmaskierte Sonderzeichen, die im XML erlaubt sind - aber im HTML nicht - ausgegeben werden sollen.

Unmaskierte Sonderzeichen, die in HTML nicht erlaubt sind, kommen in der XML Eingabe vor.

▶ Bei einer *<xsl:output> Deklaration für HTML* wird es in diesem Fall zu einem Saxon Fehler kommen.

Zeichen entfernen

Ein Lösungsweg wäre die HTML5 Deklaration zu lassen, aber die wenigen bekannten Zeichen, die in der XML Eingabe vorkommen können, auf erlaubte Entities zu mappen oder zu entfernen. Letzteres habe ich z.B. so in der freien Wildbahn gesehen:

```
<xsl:character-map name="html">
   <xsl:output-character character="&#148;" string=""/>
   <xsl:output-character character="&#149;" string=""/>
[...]
```

Das Hex Entity No. 148 ist z.B. ein spezielles Anführungszeichen[81], das im obigen Beispielquelltext einfach mittels der *Character Map* aus dem Ausgabebaum entfernt wird. Das Zeichen kann dann in einem Fehlerlog ausgegeben werden, und die Transformation läuft durch. Natürlich sollte man sich aber auch überlegen, warum ein solches Zeichen unmaskiert in die XML Eingabe gelangt ist.

M.E. ist es am besten einen XHTML Doctype zu verwenden und nicht abgeschlossene HTML5 Tags zu verbieten.

80) https://www.html-tidy.org/
81) https://www.codetable.net/decimal/148'Webreferenz zum Entity 148

Zeichen als Entity sichtbar machen

Ein anderer Use Case für so eine *Character Map* wäre, unsichtbare Zeichen sichtbar zu machen. Z.B. stellen XML Editoren, wie XMetal, auch unsichtbare Zeichen im Klartext dar. Man sieht also nicht, ob z.B. ein geschütztes Leerzeichen oder ein geschützter Bindestrich bedatet ist. Die folgende *Character Map* Deklaration würde diese Zeichen als Entität sichtbar machen - möglich ist das z.B. als Vorprozessierung der XMetal Eingabe, wenn diese Zeichen per Copy 'n Paste eingefügt werden:

```
<xsl:character-map name="desired-entities">
  <xsl:output-character character=" " string="&amp;nbsp;"/>
  <xsl:output-character character="&#x2011;" string="&amp;nbhy;"/>
</xsl:character-map>
```

Hilfreich ist in diesem Zusammenhang vielleicht auch die Möglichkeit, dass Saxon direkt Entities in den Ausgabebaum schreiben kann. Hierzu gibt es die Erweiterung `saxon:character-representation` , vgl. Saxon Doku[82)]

Exkurs

Sonderzeichen-Probleme in Browsern

Als letztes Beispiel für die Verwendung einer *Character Map* möchte ich Legacy Probleme bei der Zeichendarstellung in den verschiedenen Browsern anführen.

Bevor es den Unicodestandard gab - dieser erreichte erst 1991 die Version 1.0 - hat Adobe für bestimmte Zeichen in den Fonts Symbol und Zapf Dingbats eigene Codepunkte vergeben, die mit Unicodezeichen ggf. kollidieren. Es gibt Mapping Tabelle für Symbol[83)] und Zapf Dingbats[84)]. Einige Codepunkte in diesen Tabellen befinden sich aber in der sogenannten "Private Use Area" des Unicode Standards, die proprietären Anwendungen vorbehalten ist. Die Browser unterstützen diese PUA Zeichen unvollständig und Firefox, Chrome und Internet Explorer stellen bestimmte Zeichen fehlerhaft dar. Firefox schneidet hier am schlechtesten ab. Ich musste im Internet Archive nachschlagen, um eine gut Quelle zu finden, die dieses Problem genauer beschreibt, vgl. hier[85)].

Wir halten einmal fest:

- Das Default Character Set für den HTML5 Doctype UTF-8.
- Per XSLT werden die Zeichen normalerweise als Hex-Entities herausgeschrieben. Das ist eine Nummer, die eine Position im Unicode Standard Zeichensatz angeben sollte. Einige Glyphennummern der betreffenden Nicht-Unicode-Problemfonts passen aber nicht auf die Unicodevorgabe und die Zeichen werden fehlerhaft dargestellt.
- Die Mapping Tabellen von Adobe korrigieren das Problem aber leider unvollständig, denn hier werden bestimmte Glyphen auf einen, in Unicode vorgesehenen, Vendor-abhängigen Bereich "Private Use Area" abgebildet. Das sind c.a. 43 Zeichen, die bei Ansicht im Browser Firefox nicht gehen, leider sind darunter auch wichtige Symbole, wie Copyright, Registered und Trademark.

82) https://www.saxonica.com/html/documentation9.8/extensions/output-extras/serialization-parameters.html
83) http://www.unicode.org/Public/MAPPINGS/VENDORS/ADOBE/symbol.txt
84) Unicode Mapping Tabelle für den Zapf Dingbats Font
85) https://web.archive.org/web/20060924061009/http://ppewww.ph.gla.ac.uk/~flavell/charset/fontface-harmful.html

3.1.12 JSON mit XSLT 1.0 und Python `lxml`

Wenn man mit **Python** programmiert, hat man leider nicht **Saxon** als XSLT Prozessor zur verfügung, sondern muss sich mit dem XSLT Prozessor der lxml[86] Bibliothek begnügen. Diese basiert auf der C Bibliothek libxslt[87]

Dabei steht nur **XSLT 1.0** zur Verfügung mit den EXSLT[88] Erweiterungen.

Unter diesen Bedingungen war es nicht ganz einfach eine Transformation zu erstellen, die XML als Eingabe nimmt und sauberes JSON als Ausgabe produziert. Sicherlich ist hierfür ein mehrstufiger Prozess erforderlich.

Um genau zu sein braucht man mindestens eine Vortransformation, die die JSON Elemente als XML erzeugt und ein Postprocessing, das möglichst generisch die spitzen Klammern auf geschweifte abbildet und einige Elementnamen verwirft.

Die folgenden JSON Strukturen lassen sich leicht identifizieren:

1. **Properties**

```
"propertyName" : propertyValue
```

   Dabei kann der *propertyValue* vom Typ *String*, *Number* oder *Boolean* sein.

2. **Benannte Objekte**

```
"objektName" : { ... }
```

3. **Bennante Listen**

```
"listenName" : [ { ... }, { ... }]
```

4. **Anonyme Objekte**

```
{ }
```

In Objekten können Properties, benannte Objekte oder benannte Listen enthalten sein. In benannten Listen können nur anonyme Objekte enthalten sein. Diese Unterscheidung hat erst einmal für meinen Anwendungsfall ausgereicht.

Umgemünzt auf XSLT Regeln, kommt man auf vier Regeln die anhand einer Attributbelegung für `@json` matchen:

```
<xsl:template match="*[@json='anonymous-object']" mode="post">
  <xsl:text>{</xsl:text>
  <xsl:apply-templates select="node()|@*" mode="post"/>
  <xsl:text>}</xsl:text>
  <xsl:if test="following-sibling::*">,</xsl:if>
</xsl:template>
```

86) https://lxml.de/
87) https://gitlab.gnome.org/GNOME/libxslt
88) https://exslt.github.io/

```
<xsl:template match="*[@json='named-list']" mode="post">
  <xsl:text>"</xsl:text>
  <xsl:value-of select="name()"/>
  <xsl:text>"</xsl:text>
  <xsl:text>:[</xsl:text>
  <xsl:apply-templates select="node()|@*" mode="post"/>
  <xsl:text>]</xsl:text>
  <xsl:if test="following-sibling::*">,</xsl:if>
</xsl:template>

<xsl:template match="*[@json='named-object']" mode="post">
  <xsl:text>"</xsl:text>
  <xsl:value-of select="name()"/>
  <xsl:text>"</xsl:text>
  <xsl:text>:{</xsl:text>
  <xsl:apply-templates select="node()|@*" mode="post"/>
  <xsl:text>}</xsl:text>
  <xsl:if test="following-sibling::*">,</xsl:if>
</xsl:template>

<xsl:template match="*[@json='property']" mode="post">
  <xsl:text>"</xsl:text>
  <xsl:value-of select="name()"/>
  <xsl:choose>
    <xsl:when test="@type='boolean' or @type='integer'">
      <xsl:text>":</xsl:text>
      <xsl:value-of select="."/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:text>":"</xsl:text>
      <xsl:value-of select="."/>
      <xsl:text>"</xsl:text>
    </xsl:otherwise>
  </xsl:choose>
  <xsl:if test="following-sibling::*">,</xsl:if>
</xsl:template>
```

In einer Vortransformation in eine Variable `pre` werden die JSON Strukturen erstmal als XML aufgebaut und dann in einem Postprocessing Step `post` auf die geschweiften Klammern gemappt:

```
<xsl:template match="/">
  <root>
    <!-- first processing step -->
    <xsl:variable name="pre">
      <xsl:copy>
        <xsl:apply-templates/>
      </xsl:copy>
    </xsl:variable>
    <!-- second processing step -->
    <xsl:apply-templates select="exsl:node-set($pre)" mode="post"/>
  </root>
</xsl:template>
```

Das Aufbauen der XML-artigen JSON Struktur mit den annotierten @json Attributen, könnte z.B. so aussehen:

```
<xsl:template match="testsuite">
  <sarif-report json="anonymous-object">
    <runs json="named-list">
      <run json="anonymous-object">
        [...]
      </run>
```

```
    </runs>
    <version json="property">2.1.0</version>
  </sarif-report>
</xsl:template>
```

Mit folgender finaler Ausgabe:

```
{ "runs" : [{ ... }],
  "version": "2.1.0"
}
```

Da Python **lxml** bzw. **libxslt** anscheinend kein `xsl:output method="text"` beherscht, muss im Python Code der Textknoten eines Root-Elements ausgelesen werden. Wenn man diesen als JSON in ein Python dictionary liest und anschliessend wieder als JSON mit Indention herausschreibt, hat man auch gleich ein schönes Pretty-Printing der JSON Strukturen. Der relevante Python Code sieht dabei so aus:

```
with fileobj:
  # We register namespace functions that can be called inside the XSLT transformation
  register_stylesheet_functions()
  # Read the XSLT stylesheet
  xsl = etree.XML(open(SARIF_TRANSFORMATION_FILE, "r", encoding="utf-8").read())
  # Transform the XML to Sarif JSON and put it into the root text node
  transform = etree.XSLT(xsl)
  result = transform(tree)
  if INDENT:
    # Load into python dict and pretty print when writing back to JSON
    jsn = json.loads(result.getroot().text)
    fileobj.write(json.dumps(jsn, indent=2))
    else:
      fileobj.write(result.getroot().text)
```

Das vollständige Beispiel findet man in meinem Github Repo[89].

3.2 Abfragen mit XQuery

Xquery führt im Publishing-Bereich ein Schattendasein. In meiner Zeit als XSL Programmierer für zwei Publishing Firmen hatte ich damit nie zu tun. Erst als ich näher an den eigentlichen Daten war und mit XML Datenbanken zu tun hatte, kam ich mit XQuery in Berührung.

Während relationale Datenbanken mit SQL abgefragt werden, verwendet man bei XML Datenbanken, wie eXist[90] oder Marklogic[91], XQuery als Abfragesprache.

Aber auch einzelne XML Dokumente können z.B. in Oxygen XML Editor mit dem XQuery Builder Tool[92] oder auch per Saxon Kommandozeile abgefragt werden:

```
java -cp usr/lib/saxon/saxon.jar net.sf.saxon.Query
     -s:"schulen.xml"
     -qs:"/schulen/schule[id='6']"
```

89) https://github.com/alexdd/bandit-sarif/blob/feature-sarif-formatter-with-cwe-extension/bandit/formatters/sarif/sarif.xsl
90) http://exist-db.org/exist/apps/homepage/index.html
91) https://de.marklogic.com/
92) https://www.oxygenxml.com/xml_editor/xquery_builder.html

```
    -o:"/Users/Alex/Desktop/schule_6.xml"
```

Mit der Option `-qs` kann hier der Querystring angebenen werden.

Wie man an dem einfachen Beispiel schon sieht, ist XQuery mit XPath verwandt. XQuery umfasst den Sprachumfang von XPath bietet aber zusätzlich die FLOWR Syntax um mächtigere Abfragen stellen zu können. Mittels weiterer Extensions[93] können aber auch ganze Programme erstellt werden, die weit über die Funktionalität einer "Abfragesprache" hinausgehen.

XQuery Builder

Oxygen XML Editor[94] bietet eine schöne Möglichkeit XQuery-Abfragen auf einem geladenen XML Dokument auszuführen. Dazu kann man seine Query in das betreffende Eingabefenster schreiben.

Mit dem XQuery Builder von oXygen lassen sich unkompliziert Queries testen

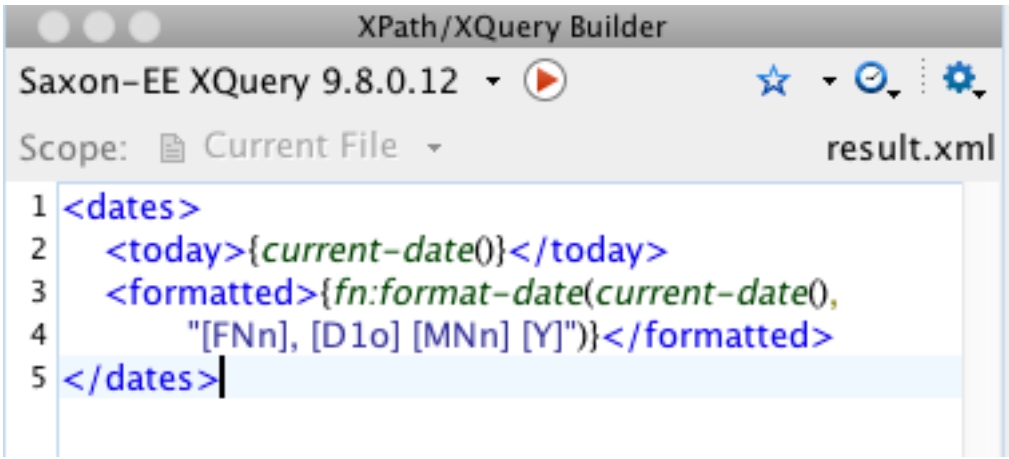


figure 11: oXgen XQuery Builder

Mit folgendem Ergebnis:

```
<dates>
  <today>2019-01-16+01:00</today>
  <formatted>Wednesday, 16th January 2019</formatted>
</dates>
```

FLOWR Expression

FLOWR steht für `for, let, where, order by, return` . Das sind die Query-Anweisungen, die in dem Ausdruck erlaubt sind - in genau dieser Reihenfolge.

```
let $bibliothek := .
for $xin $bibliothek//buecher,
    $yin $bibliothek//autoren/autor
where starts-with($autor, 'Grass')
    and $x/@autorId = $y/@id
return $x/titel
```

In dieser Query werden die Titel aller Bücher von Grass zurückgeliefert. Bemerkenswert ist hier die Syntax.

93) http://cs.au.dk/~amoeller/XML/querying/flwrexp.html
94) https://www.oxygenxml.com/xml_editor

▻ Normalerweise würde man zwischen den einzelnen Anweisungen einen Blockabschluss, wie ein Semikolon erwarten. Da wir aber hier funtional prorgammieren, ist die Sache etwas anders...

XML per XQuery

Es ist aber auch möglich XML zu erzeugen, wobei natürlich für eine Transformation XSLT vorzuziehen ist. Dazu werden Tags direkt in die Expression geschrieben, wie z.B. hier:

```
declare variable $nachname as xs:string external;
<buecher autor="{$nachname}">
{
  let $bibliothek := .
  for $xin $bibliothek/buecher//buch,
      $yin $bibliothek/autoren//autor
  where starts-with($y, $nachname)
      and $x/@autorId = $y/@id
  order by $x/ausgabe
  return
  <buch ausgabe="{$x/ausgabe}">
    {$x/titel}
  </buch>
}
</buecher>
```

Speichert man dieses Schnippsel in einer Datei *buecher.xquery* ab, so kann man mit der folgenden Kommandozeile auf einer *buecher.xml* Datei als Eingabe suchen:

```
java -cp usr/lib/saxon/saxon.jar net.sf.saxon.Query -t -s:buecher.xml
                                                       -q:buecher.xquery
                                                       -o:ergebnis.xml
                                                       nachname=grass
```

Document Projection

Document Projection[95] ist ein verstecktes Saxon XQuery Feature. Es funktioniert nur für eine einzige Abfrage. Das kann schon recht hilfreich sein, wenn man ein mehrere 100MB großes Dokument durchsuchen will.

Ohne Projection würde das Beispiel von oben so verarbeitet:

```
java -cp usr/lib/saxon/saxon.jar net.sf.saxon.Query -t
     -s:buecher.xml
     -q:buecher.xquery
     -o:ergebnis.xml
     -projection:off
     nachname=grass
Saxon-EE 9.7.0.20J from Saxonica
Java version 1.8.0_60
Using license serial number V005095
Analyzing query from Desktop/buecher.xquery
Generating byte code...
Analysis time: 201.10095 milliseconds
Processing file:/Users/Alex/buecher.xml
Using parser com.sun.org.apache.xerces.internal.jaxp.SAXParserImpl$JAXPSAXParser
Building tree for file:/Users/Alex/buecher.xml
using class net.sf.saxon.tree.tiny.TinyBuilder
Tree builtin 3.482278ms
Tree size: 46 nodes, 58 characters, 6 attributes
```

95) http://www.saxonica.com/documentation/#!sourcedocs/projection

```
Execution time: 27.137589ms
Memory used: 67031664
```

Mit der Option `-projection:on` verändert sich die Ausführungszeit signifikant:

```
[...]

Document projection for file:/Users/Alex/buecher.xml
-- Input nodes 50; output nodes 27; reduction = 46%
Tree builtin 3.80615ms
Tree size: 26 nodes, 58 characters, 3 attributes
Execution time: 15.83463ms
Memory used: 64339064
```

3.2.1 XQuery als Programmiersprache

Erste Schritte in XQuery gehen sehr schön mit der Query Konsole auf dem Marklogic Server (Port 8000). Hat man parallel auch noch oXygen Editor offen, kann man die Testergebnisse aus der Konsole direkt in ein XQuery Server Skript packen.

Schleifen

Die ersten 10 Dokumente auf dem Server bekommt man z.B. mit:

```
(doc())[position() lt 11]
```

Die Zahlen von 1 bis 100 mit:

```
(for $iin (1 to 100) return $i)
```

Wenn man diese beide Anweisungen untereinander in die Konsole schreibt bekommt man einen Fehler. Trennt man sie mit einem Komma - ein Tupel wird erzeugt - dann klappt es.

Dokumente in der DB anlegen

Der Befehl zum Anlegen eines Dokuments in der Marklogic DB sieht folgendermassen aus (Doku)[96]:

```
xdmp:document-insert(
  "/alex-test/example-alex.xml",
  <root>Hier steht der Content</root>,
  <options xmlns="xdmp:document-insert">
    <metadata>{
      map:map() => map:with("valid-start", "2014-06-03T14:13:05.472585-07:00")
      => map:with("valid-end", "9999-12-31T11:59:59Z")
      }</metadata>
  </options>)
```

Packt man diese Instruktion in die for-Schleife oben, dann sieht das Konstrukt so aus:

```
(for $iin (1 to 10) return
  xdmp:document-insert(
  concat("/alex-test/example-alex-",$i,".xml"),
  <root>Hier steht der Content {$i}</root>,
  <options xmlns="xdmp:document-insert">
    <metadata>{
```

96) https://docs.marklogic.com/xdmp:document-insert

```
      map:map() => map:with("valid-start", "2014-06-03T14:13:05.472585-07:00")
      => map:with("valid-end", "9999-12-31T11:59:59Z")
      }</metadata>
  </options>)
)
```

Dokumente kann man einer Collection zuweisen, um sie leichter finden und auswerten zu können. Das geht mit dem folgenden Befehl (Doku)[97] und diesem Schnippsel:

```
let $root :=
<mein-test>
   <id>{$id}</id>
   <content>Hallo Welt!</content>
</mein-test>,
$options :=
<options xmlns="xdmp:document-insert">
  <permissions>{xdmp:default-permissions()}</permissions>
  <collections>
    <collection>/alex-test</collection>
  </collections>
</options>,
$fname := concat('/',$id,'_','.xml'),
$td := xdmp:document-insert($fname,$root,$options)
return
  [...]
```

Die Dokumente, die mit der Collection `alex-test` getaggt wurden, kann man sich mit der folgenden Schleife ausgeben lassen:

```
for $xin collection("/alex-test")
  return
    fn:document-uri($x)
```

3.2.1.1 `if..then..else` Ausdrücke

In nicht-funktionalen Programmiersprachen sind die Schlüsselwörter `if` und `then` dazu da, um dem Compiler oder Interpreter mitzuteilen, dass eine bedingte Anweisung ausgewertet werden soll.

Was für den Nicht-funktionalen Programmierer etwas befremdlich erscheint, ist der Umstand, dass in XQuery `if..then` als Ausdrücke ausgewertet werden.

Das ist einerseits sehr praktisch, weil es richtig angewandt den Code verkürzt und damit das Wesentliche herausstellt, kann aber auch weiter zur allg. Verwirrung bzgl. des kryptischen XQuery Codes beitragen.

Beispiel: Konditionale Server App

Betrachten wir ein einfaches Beispiel: Wir generieren auf einer Marklogic-Webapp eine JSON Response. Da wir diesen Mechanismus an mehreren Stellen im Code einsetzen, empfiehlt es sich das Rendern des Headers in eine Funktion auszulagern.

```
declare function common:wrap-response($json)
{
  xdmp:add-response-header("Pragma", "no-cache"),
  xdmp:add-response-header("Cache-Control", "no-cache"),
  xdmp:add-response-header("Expires", "0"),
  xdmp:set-response-content-type('text/json; charset=utf-8'),
  $json
```

97) https://docs.marklogic.com/xdmp:document-add-collections

```
};
```

In unserem Request-Handler wird je nach Auswertung einer Variablen eine bedingte Anweisung ausgeführt, diese sieht bspw. so aus:

```
let $name := xdmp:get-request-field('name'),
    $is-afternoon := xs:time(current-dateTime()) gt xs:time('12:00:00')
return
  if ($is-afternoon) then
   common:wrap-response(xdmp:unquote(concat('{"greeting":"Good Afternoon! ',$name,'!"}')))
  else
   common:wrap-response(xdmp:unquote(concat('{"greeting":"Good Morning! ',$name,'!"}')))
```

Als prozeduraler Programmierer wäre ich mit diesem Switch voll und ganz zufrieden, der funktionale Programmierer erkennt aber sofort einen Optimierungsbedarf.

Da es sich bei der bedingten Anweisung auch um einen Ausdruck handelt, der `true` oder `false` zurückgibt, können wir die gleichen Funktionsaufrufe herausziehen:

```
ommon:wrap-response(xdmp:unquote(
    let $is-afternoon := xs:time(current-dateTime()) gt xs:time('12:00:00')
        return
            if ($is-afternoon) then concat('{"greeting":"Good Afternoon! ',$name,'!"}')
            else concat('{"greeting":"Good Morning! ',$name,'!"}')
)
```

Hier wird der abstrakt denkende Programmierer aber einwenden, dass eine abstrakte Logik nicht in eine Low-Level Funktion, wie `xdmp:unquote` gewrapped werden sollte.

Das stimmt - und mehr noch, die Maskierung mit `xdmp:unquote` sollte auch noch in unsere Funktion gepackt werden. So dass der Code schliesslich so aussehen würde:

```
declare function common:render-response($json)
{
  xdmp:add-response-header("Pragma", "no-cache"),
  xdmp:add-response-header("Cache-Control", "no-cache"),
  xdmp:add-response-header("Expires", "0"),
  xdmp:set-response-content-type('text/json; charset=utf-8'),
  xdmp:unquote($json)
};
common:render-response(
    let $is-afternoon := xs:time(current-dateTime()) gt xs:time('12:00:00')
        return
            if ($is-afternoon) then concat('{"greeting":"Good Afternoon! ',$name,'!"}')
            else concat('{"greeting":"Good Morning! ',$name,'!"}')
)
```

Sicherlich lässt sich darüber streiten, ob nun der funktionale Ansatz besser lesbar ist, oder der prozedurale von oben.

Ich denke jeder Programmierer hat hier seinen eigenen, individuellen und bewährten Programmierstil entwickelt, den er auch beibehalten sollte.

SQL Views in MarkLogic

Es macht nicht immer Sinn über eine Baumstruktur zu suchen. Obwohl das in einer XML Datenbank rasend schnell geht, weil jeder Knoten des Baums initial in einen Index

aufgenommen wird. So gibt es doch Anwendungsfälle bei denen man lieber eine relationale Sicht auf die Daten hätte.

In MarkLogic heisst die Lösung dazu SQL Views.

Bspw. benötigt man eine relationale Sicht auf die Daten, wenn über verschiedene Datensätze ein Report generiert werden soll.

Nehmen wir an, es gibt im D- die folgenden Dokumente:

```
<k:kunde>
  <k:id>1</k:id>
  <k:name>Alex</k:name>
  <k:eMail>tekturcms@gmail.com</k:eMail>
</k:kunde>

<k:kunde>
  <k:id>2</k:id>
  <k:name>Horst</k:name>
  <k:eMail>horst@horst.de</k:eMail>
</k:kunde>

<k:kunde>
  <k:id>3</k:id>
  <k:name>Gundula</k:name>
  <k:eMail>gundl@gundula.de</k:eMail>
</k:kunde>

<b:bestellung>
  <b:id>1</b:id>
  <b:datum>02.01.2019</b:datum>
  <b:preis>99.90</b:preis>
  <kunde-id>2</kunde-id>
</b:bestellung>

<b:bestellung>
  <b:id>2</b:id>
  <b:datum>03.01.2019</b:datum>
  <b:preis>68.90</b:preis>
  <b:kunde-id>1</b:kunde-id>
</b:bestellung>
```

Will man sich alle Kunden anzeigen lassen, die eine Bestellung abgegeben haben - das sind Alex und Horst - so würde man bei einem relationalen Ansatz einen `JOIN` verwenden, so wie hier:

```
SELECT name, datum, preis
FROM kunden k
INNER JOIN bestellungen b
ON k.id = b.kunde_id
```

In einer relationalen Sicht würde uns das dann die folgende Tabelle liefern:

```
name, datum, preis
Alex, 03.01.2019, 68.90
Horst, 02.01.2019, 99.90
```

Um für MarkLogic eine SQL View zu definieren verwendet man einen Mechanismus, der da heisst: Template Driven Extraction[98)]

98) https://docs.marklogic.com/guide/app-dev/TDE

Dazu werden Templates in XML deklariert und in die Template Collection eingefügt. Für unser obiges Beispiel würden wir zwei Templates brauchen, die so aussehen:

```
xquery version "1.0-ml";

import module namespace tde = "http://marklogic.com/xdmp/tde"
                              at "/MarkLogic/tde.xqy";

let $sql-view-name := 'kunden-view.xml'
let $sql-view := <template xmlns="http://marklogic.com/xdmp/tde">
   <path-namespaces>
      <path-namespace>
         <prefix>k</prefix>
         <namespace-uri>https://tekturcms.de/schema/kunde/1.0</namespace-uri>
      </path-namespace>
   </path-namespaces>
   <context>/k:kunde</context>
   <collections>
      <collections-and>
         <collection>/kunden</collection>
      </collections-and>
   </collections>
   <rows>
      <row>
         <schema-name>kunden_schema</schema-name>
         <view-name>kunden_view</view-name>
         <columns>
            <column>
               <name>id</name>
               <scalar-type>string</scalar-type>
               <val>k:id</val>
               <nullable>true</nullable>
            </column>
            <column>
               <name>datum</name>
               <scalar-type>string</scalar-type>
               <val>k:datum</val>
               <nullable>true</nullable>
            </column>
            <column>
               <name>eMail</name>
               <scalar-type>string</scalar-type>
               <val>k:eMail</val>
               <nullable>true</nullable>
            </column>
         </columns>
      </row>
   </rows>
</template>
return(
  tde:template-insert(concat('/templates/', $sql-view-name),
                      $sql-view, xdmp:default-permissions())
)
```

und analog für die Bestellungen:

```
[...]
   <rows>
      <row>
         <schema-name>bestellungen_schema</schema-name>
         <view-name>bestellungen_view</view-name>
         <columns>
            <column>
               <name>id</name>
```

```
               <scalar-type>string</scalar-type>
               <val>b:id</val>
            </column>
            <column>
               <name>datum</name>
               <scalar-type>string</scalar-type>
               <val>b:datum</val>
            </column>
            <column>
               <name>preis</name>
               <scalar-type>string</scalar-type>
               <val>b:preis</val>
            </column>
            <column>
               <name>kunde_id</name>
               <scalar-type>string</scalar-type>
               <val>b:kunde-id</val>
            </column>
         </columns>
      </row>
[...]
```

In XQuery eingebunden, könnte man dann die definierten SQL Views mit dem folgenden Befehl abfragen:

```
xdmp:sql("SELECT name, datum, preis FROM kunden_view k
          INNER JOIN bestellungen_view b ON k.id = b.kunde_id")
```

Folgende ist das komplette Beispiel für eine MarkLogic XQuery Konsolensitzung abgebildet ...

```
xquery version "1.0-ml";

declare namespace k = 'http://www.tekturcms.de/kunden';
declare namespace b = 'http://www.tekturcms.de/bestellungen';

import module namespace tde = "http://marklogic.com/xdmp/tde" at "/MarkLogic/tde.xqy";

declare function local:loadKunde($id, $name, $eMail)
{
  let $root :=
  <k:kunde>
    <k:id>{ $id }</k:id>
    <k:name>{ $name }</k:name>
    <k:eMail>{ $eMail }</k:eMail>
  </k:kunde>,
  $options :=
  <options xmlns="xdmp:document-insert">
    <permissions>{ xdmp:default-permissions() }</permissions>
    <collections>
      <collection>/kunden</collection>
    </collections>
  </options>,
  $fname := concat('/kunden/',$id,".xml")
  return xdmp:document-insert($fname,$root,$options)
};

declare function local:loadBestellung($id, $datum, $preis, $kunde-id)
{
  let $root :=
  <b:bestellung>
    <b:id>{ $id }</b:id>
    <b:datum>{ $datum }</b:datum>
```

```
    <b:preis>{ $preis }</b:preis>
    <b:kunde-id>{ $kunde-id }</b:kunde-id>
  </b:bestellung>,
  $options :=
  <options xmlns="xdmp:document-insert">
    <permissions>{ xdmp:default-permissions() }</permissions>
    <collections>
      <collection>/bestellungen</collection>
    </collections>
  </options>,
  $fname := concat('/bestellungen/',$id,".xml")
  return xdmp:document-insert($fname,$root,$options)
};

declare function local:insertKundenSchema()
{
  let $sql-view-name := 'kunden-view.xml',
      $sql-view := <template xmlns="http://marklogic.com/xdmp/tde">
    <path-namespaces>
      <path-namespace>
        <prefix>k</prefix>
        <namespace-uri>http://www.tekturcms.de/kunden</namespace-uri>
       </path-namespace>
    </path-namespaces>
    <context>/k:kunde</context>
     <collections>
       <collections-and>
         <collection>/kunden</collection>
      </collections-and>
     </collections>
     <rows>
       <row>
         <schema-name>kunden_schema</schema-name>
         <view-name>kunden_view</view-name>
         <columns>
            <column>
               <name>id</name>
               <scalar-type>string</scalar-type>
               <val>k:id</val>
            </column>
            <column>
               <name>name</name>
               <scalar-type>string</scalar-type>
               <val>k:name</val>
            </column>
          <column>
             <name>eMail</name>
             <scalar-type>string</scalar-type>
             <val>k:eMail</val>
          </column>
        </columns>
      </row>
    </rows>
  </template>
  return
    tde:template-insert(concat('/templates/',
                        $sql-view-name), $sql-view, xdmp:default-permissions())
};

declare function local:insertBestellungenSchema()
{
  let $sql-view-name := 'bestellungen-view.xml',
      $sql-view := <template xmlns="http://marklogic.com/xdmp/tde">
    <path-namespaces>
      <path-namespace>
        <prefix>b</prefix>
        <namespace-uri>http://www.tekturcms.de/bestellungen</namespace-uri>
```

```
      </path-namespace>
    </path-namespaces>
    <context>/b:bestellung</context>
     <collections>
       <collections-and>
         <collection>/bestellungen</collection>
      </collections-and>
     </collections>
     <rows>
       <row>
         <schema-name>bestellungen_schema</schema-name>
         <view-name>bestellungen_view</view-name>
         <columns>
            <column>
               <name>id</name>
               <scalar-type>string</scalar-type>
               <val>b:id</val>
            </column>
            <column>
               <name>datum</name>
               <scalar-type>string</scalar-type>
               <val>b:datum</val>
            </column>
          <column>
             <name>preis</name>
             <scalar-type>string</scalar-type>
             <val>b:preis</val>
          </column>
          <column>
             <name>kunde_id</name>
             <scalar-type>string</scalar-type>
             <val>b:kunde-id</val>
          </column>
        </columns>
      </row>
    </rows>
  </template>
  return
    tde:template-insert(concat('/templates/',
                        $sql-view-name), $sql-view, xdmp:default-permissions())
};

local:loadKunde("1","Alex","tekturcms@gmail.com"),
local:loadKunde("2","Horst","horst@horst.de"),
local:loadKunde("3","Gundula","gundl@gundula.de"),
local:loadBestellung("1","02.01.2019","99.90","2"),
local:loadBestellung("2","03.01.2019","68.90","1"),
local:insertKundenSchema(),
local:insertBestellungenSchema(),
xdmp:sql("SELECT name, datum, preis FROM kunden_view k INNER JOIN
          bestellungen_view b ON k.id = b.kunde_id")
```

... mit einer schönen tabellarischen Ausgabe im unteren Panel der Query Konsole - oder als JSON:

```
[
  [
    "k.name",
    "b.datum",
    "b.preis"
  ],
  [
    "Alex",
    "03.01.2019",
    "68.90"
```

```
  ],
  [
    "Horst",
    "02.01.2019",
    "99.90"
  ]
]
```

3.2.2 Hilfreiche XQuery Schippsel

Unsortierte Schnippsel für die MarkLogic Konsole, die an mancher Stelle hilfreich sein könnten:

Beschreibung	Beispiel Code Schnippsel

Datenbank löschen	`xdmp:forest-clear(`
	`xdmp:database-forests(`
	`xdmp:database("xml-scrapper-content")))`

Über Collection iterieren	`for $xin collection("/topics")`
	`return`
	` (<fname>{ fn:document-uri($x) }</fname>,`
	` $x)`

Attribut / Knoten ersetzen	`for $refin collection("/topics")//topicref`
	`  let $topic := collection("/topics")`
	`  [contains(.),$ref/@href)],`
	`  $new := attribute href { $topic/topic/@id }`
	`return`
	`  xdmp:node-replace($ref/@href,  $new)`

Dokument "umbenennen"[1)]	`xdmp:document-insert(`
	`  $new-uri,`
	`  fn:doc($old-uri),`
	`  xdmp:document-get-permissions($old-uri),`
	`  xdmp:document-get-collections($old-uri)`
	` ),`
	` xdmp:document-delete($old-uri),`
	` xdmp:document-set-properties($new-uri, $properties`

Über verschiedene Collections bestimmte Elemente suchen	`for $elementin cts:search(/descendant::*[self::b or`
	`  self::i or self::u],`
	`  cts:collection-query(("/topics",`
	`  "/tasks", "/maps")))`
	`return $element/text()`

Ein Element aus einem "Stack" holen und das Element vom Stack nehmen ("Pop")	`let $result := (`
	`  for $stackin collection("/stack-pushed")`
	`  order by $stack/some-element/some-criterion descending`
	`  return`
	`  $stack)[1]`
	`return (`
	`  local:render-response($result),`
	`  xdmp:document-set-collections(fn:document-uri($result),`
	`  ("/stack-popped")`

1) https://developer.marklogic.com/recipe/move-a-document/

Konvertierung nach JSON	`let $xml:=`
	`<tektur-dita>`
	`{`
	`for $xin collection("/topics")`
	`return`
	` $x`
	`}`
	`</tektur-dita>,`
	`  $config := json:config("full") =>`
	`  map:with("whitespace", "preserve")`
	`  return json:transform-to-json( $xml , $config )`

Gib mir alle Dokumente, die nach dem 23.05.2019 zur (temporalen) Collection "/topics" hinzugefügt wurden	`let $period := cts:period(`
	`  xs:dateTime('0001-01-01T00:00:00'),`
	`  xs:dateTime('2019-05-23T00:00:00')`
	`)`
	`return cts:search(`
	`  collection("/topics"),`
	`  cts:period-range-query('system-axis',`
	`  'ISO_SUCCEEDS',`
	`  $period)`
	`)`

cURL Parameter um eine DB zu leeren (als Array Einträge im Python Code)	`args = [`
	`  'curl',`
	`  '-X',`
	`  'POST',`
	`  '--anyauth',`
	`  '-u',`
	`  'admin:admin',`
	`  '--header',`
	`  'Content-Type:application/json',`
	`  '-d',`
	`  '{\"operation\":\"clear-database\"}',`
	`  'http://localhost:8002/manage/`
	`  v2/databases/mydb']`

Elementare Permissions an Dokumenten setzen; **xdmp:document-update-permissions** zum Aktualisieren	`for $x in collection("/my-collection")`
	`return`
	` xdmp:document-set-permissions(`
	`  fn:document-uri($x),`
	`  ( xdmp:permission("my-role", "update"),`
	`  xdmp:permission("my-role", "read" )))`

3.3 XML Datenbanken

XML Datenbanken konzentrieren sich im Gegensatz zu den verbreiteten relationalen Datenbanken auf die Struktur eines Dokuments, die abstrakt gesehen einen Baum darstellt, und weniger auf die Beziehungen zwischen Objekten, die eher einen Graphen aufspannen.

Natürlich ist auch jeder Baum ein Graph ohne Kreise, und sicherlich kann man auch Bäume in einer relationalen Datenbank abspeichern. Eine XML Datenbank ist aber für diese Struktur optimiert.

Es gibt gegenwärtig vier reine XML Datenbanken und einige Erweiterungen für konventionelle SQL Datenbanken:

Datenbank	Besonderheiten
eXist DB [EX]	eXist DB ist ein Open Source Projekt. Neben der Datenbank umfasst diese Software eine komplette Entwicklungsumgebung für Webapplikationen. Für diese DB existiert ein 1-Klick Installer in Form eines Java Jars. eXist ist im Bereich Digital Humanities (ein Fachbereich der Geschichts- und Kulturwissenschaften) sehr verbreitet.
BaseX [BX]	BaseX ist ebenfalls OpenSource und die Homepage macht einen ordentlichen Eindruck. Bisher bin ich noch nicht dazugekommen, BaseX zu evaluieren. Unter meiner Java 10 Installation lief erst einmal nichts - weshalb die erste Kontaktaufnahme scheiterte.
MarkLogic [ML]	MarkLogic ist der Platzhirsch unter den kommerziellen Anbietern. Hier ist alles "Enterprise"... die Funktionalität, der Support und auch der Preis. Obwohl ML viele Erweiterungen für XQuery bietet, ist der XQuery 3.0 Standard noch nicht umgesetzt.
Berkely DB XML Datenbank [BD]	Die gute alte Berkely DB war der Key-Value Unterbau für viele andere Datenbanken, wie auch MySQL. Sicherlich hat auch die XML Variante einiges in Petto.

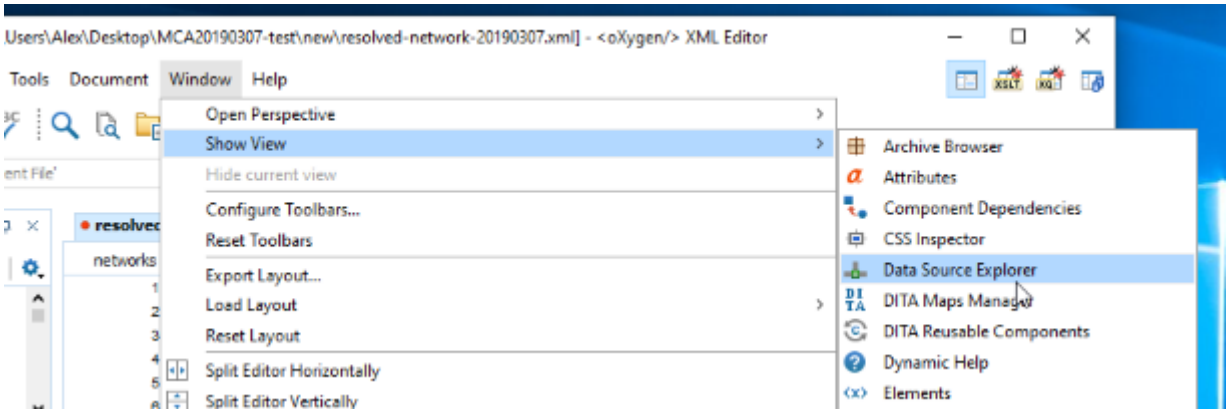
Da ich zur Zeit beruflich mit MarkLogic zu tun habe, lasse ich mir die Gelegenheit nicht nehmen, meine Erfahrungen und Erkenntnisse dazu in diesem Kapitel zu beschreiben. Es gibt auch eine Developer License [DL] mit der man die Software ausprobieren kann. Für alle langfristigen XQuery Spielereien ist die eXist DB wohl die erste Wahl, da hier auch der aktuelle XQuery Standard umgesetzt ist.

3.3.1 Connector zu Marklogic in Oxygen

Marklogic bietet zwar auf Port 8000 per Default ein Query Console im Browser, mit der man bestimmte Sachen ausprobieren kann. Komfortabler arbeitet man aber mit einem Oxygen-Connector . Hier öffnet man den *Data Source Explorer* und konfiguriert eine neue Datenquelle:

[EX] http://exist-db.org/exist/apps/homepage/index.html
[BX] http://basex.org/
[ML] https://www.marklogic.com/
[BD] https://www.oracle.com/database/berkeley-db/xml.html
[DL] https://developer.marklogic.com/free-developer

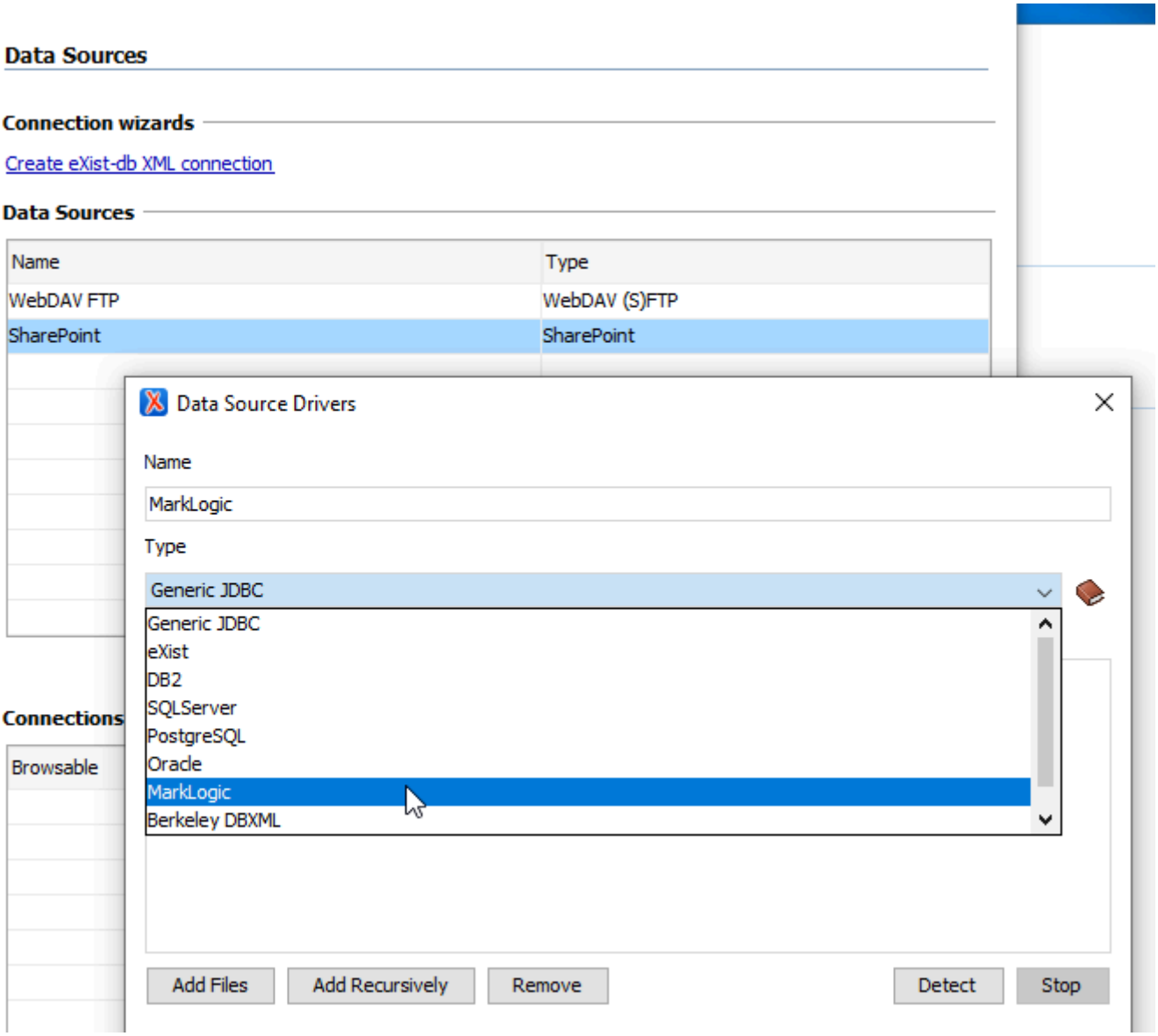
oXygen Datasource Explorer View öffnen



Dazu muss man den Marklogic Treiber installieren[105)] und diesen im folgenden Screen verfügbar machen.

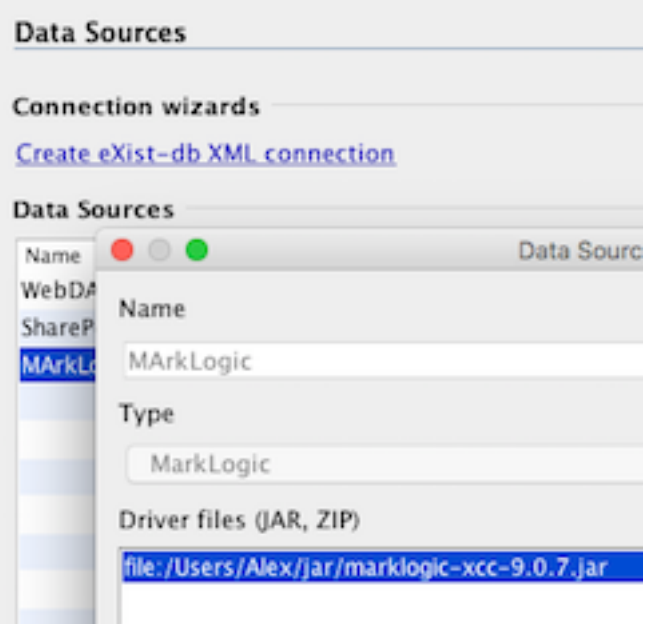
105) https://www.oxygenxml.com/doc/versions/20.1/ug-editor/topics/configure-marklogic-datasource.html

Neue Datenquelle in oXygen konfigurieren



Das Jar sollte an einem soliden Ort abgespeichert werden, da hier nur ein Verweis auf diesen Ort gesetzt wird.

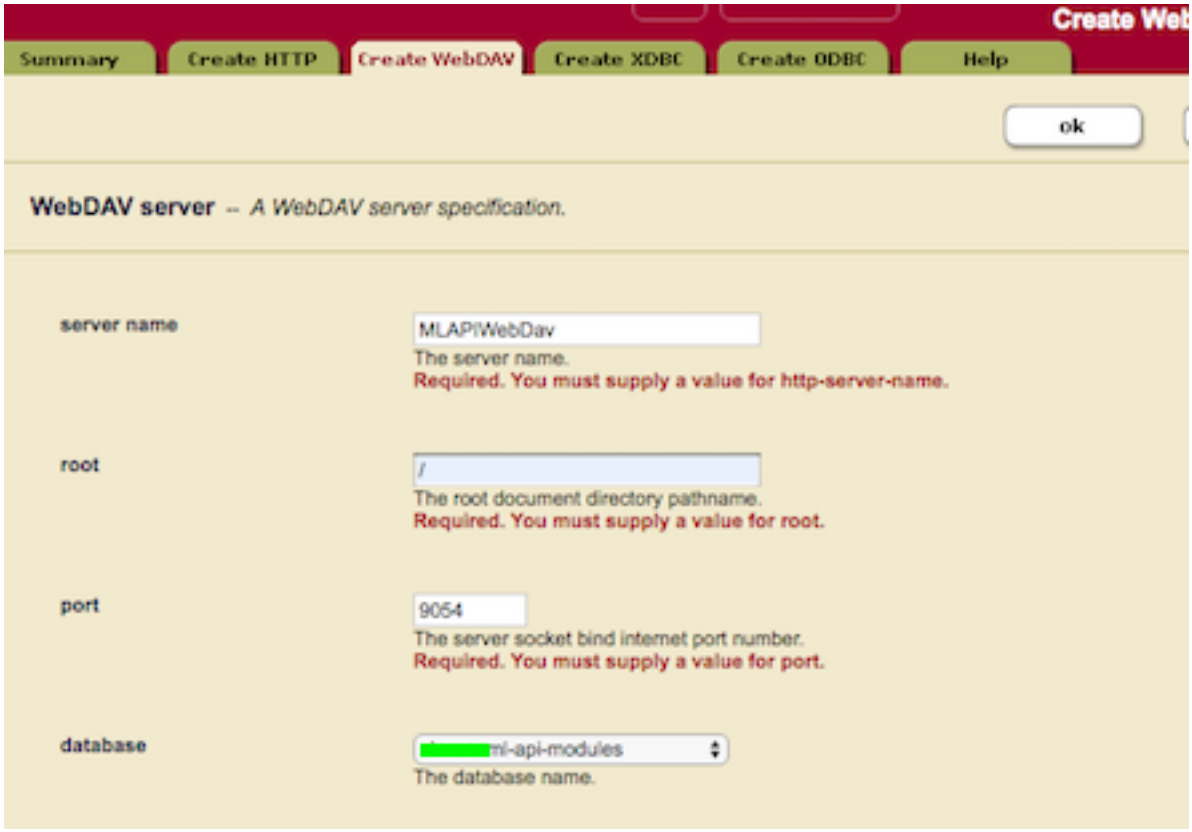
Neue oXygen Treiberdatei auswählen



Natürlich ist auf der Serverseite auch eine Einstellung notwendig. Man wechselt als Admin in den Bereich **App Servers** und fügt einen neuen **WebDAV Server** hinzu. Ggf. muss man bei der Auswahl der Datenbank diese noch auf "automatische Directory Erzeugung" umstellen.

Wechseln in die MarkLogic Appserver Verwaltung

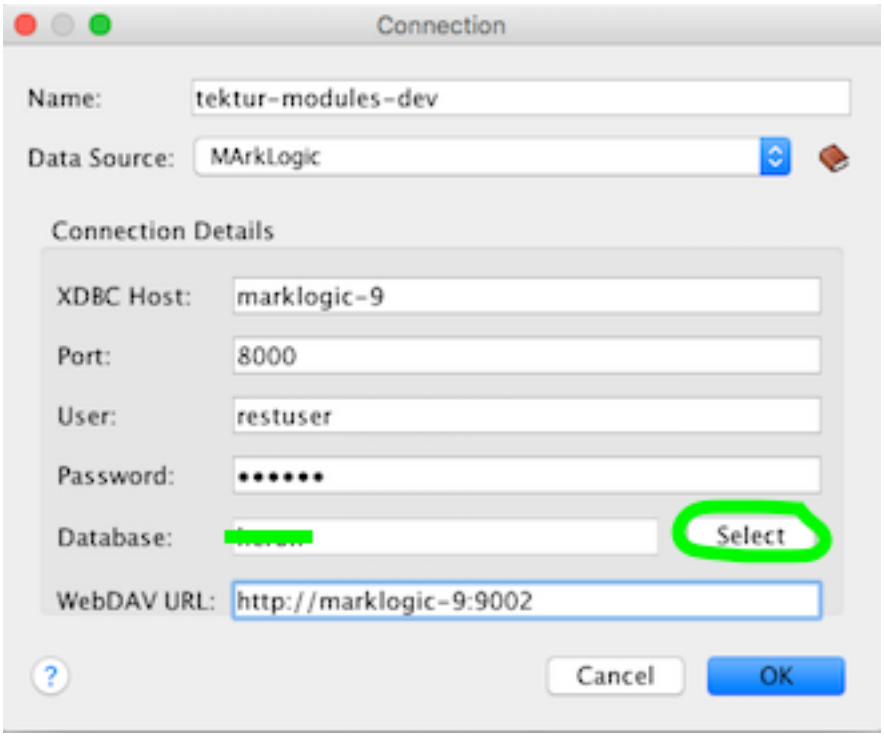
WebDav in MarkLogic konfigurieren



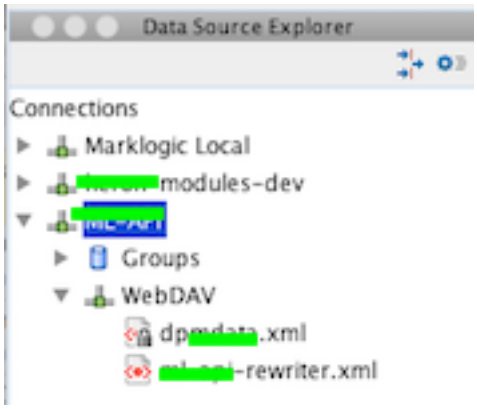
Hat man den WebDAV Server erzeugt und diesem eine bestehende oder neu angelegte Datenbank zugewiesen, dann kann man diese DB im Konfigurationsdialog der neuen WebDav Connection auswählen.

Schliesslich hat man im Data Source Explorer in oXygen die neue Verbindung verfügbar und kann gefühlt wie im Dateisystem mit den Files auf dem Server arbeiten.

Konfigurieren der WebDAV Connection Einstellungen in oXygen



oXygen Data Source Explorer zeigt die WebDAV Verzeichnisse auf dem Marklogic Server



3.3.2 Bi-Temporale Dokumente

Wenn wir zwischen der Zeit in der das Dokument, bspw. ein Vetrag in der DB angelegt wird, und der Zeit in der ein Vetrag zwischen zwei Vetragspartnern abgeschlossen wird unterscheiden, dann betrachten wir zwei Zeitachsen.

- die gültige Zeit (Valid Time)
- die Systemzeit (System Time)

Für die Vetragspartner ist nur die gültige Zeit relevant. Das Zeitfenster zwischen gültiger Zeit und Systemzeit ist jedoch in manchen Fällen ausschlaggebend.

Bsp: Kürzlich wurde meine Geldbörse mit meinem Perso geklaut. Beim Austellen eines vorläufigen Ausweises wurde ich schriftlich darauf hingewiesen, dass nun mein Perso bei Interpol zur Fahndung ausgeschrieben ist.

Kurze Zeit später fand ein netter Herr die Geldbörse (ohne Geld aber mit allen Papieren) in seinem Garten. Bei einer anschliessenden Busfahrt mit einem Fernbus, wurde ich bei einer Zollkontrolle festgehalten, da das System der Polizei noch nicht aktualisiert war.

Ich nehme an, dass nach meiner Unschuldsbekundung der Vorgang auch auf Seiten des Polizeicomputers aktualisiert wurde. Nun könnte man zwei Fragen stellen:

1. Ist das Festhalten seitens der Zollbeamten rechtens?
2. Habe ich mich durch ein verspätetes Anzeigen des Funds schuldig gemacht?

> ▻ Beachte, dass man diese Fragen auch noch nach 10 Jahren stellen könnte und das - bei meinem Pech in diesen Angelegenheiten - so ein Vorfall auch noch öfters passieren könnte...

Um diese Fragen zu beantworten, müsste unsere Datenbank in der Lage sein, eine bitemporale Query[106] auszuführen. Zunächst registerieren wir den Vorgang des Persoverlustes in unserer Marklogic Datenbank:

> ▻ Da wir hier auf einer XML Datenbank arbeiten, sprechen wir von einem Dokument, wenn wir einen Datensatz meinen.
>
> Der Datensatz bzw. das Dokument wird nicht aktualisiert, sondern stattdessen das Dokument mit den aktualisierten Daten in einer neuen Version angelegt.
>
> Auf diese Weise bleibt die Änderungshistorie erhalten.

```
xquery version "1.0-ml";
import module namespace temporal =
       "http://marklogic.com/xdmp/temporal" at "/MarkLogic/temporal.xqy";
let $root :=
  <vorgang>
    <perso-id>XYZ</perso-id>
    <name>Alex Düsel</name>
    <status>gestohlen</status>
  </vorgang>
let $options :=
  <options xmlns="xdmp:document-insert">
    <metadata>
      <map:map xmlns:map="http://marklogic.com/xdmp/map">
        <map:entry key="validStart">
   <map:value>2019-02-01T08:23:11</map:value>
 </map:entry>
 <map:entry key="validEnd">
   <map:value>9999-12-31T11:59:59Z</map:value>
 </map:entry>
      </map:map>
    </metadata>

return  temporal:document-insert("/perso-verluste",
                                 "duesel_alex_270774.xml",
                                 $root, $options)
```

106) https://en.wikipedia.org/wiki/Temporal_database

Unser Enddatum liegt in ferner Zukunft sicherzustellen, dass der Vorgang auf unbestimmte Zeit im System bleibt.

Drei Tage später hatte ich meinen Ausweis wieder und der Vorgang wurde vier Tage später, mit dem Status "gefunden" im Polizeicomputer aktualisiert:

```
xquery version "1.0-ml";
import module namespace temporal =
       "http://marklogic.com/xdmp/temporal" at "/MarkLogic/temporal.xqy";
let $root :=
  <vorgang>
    <perso-id>XYZ</perso-id>
    <name>Alex Düsel</name>
    <status>gefunden</status>
  </vorgang>
let $options :=
  <options xmlns="xdmp:document-insert">
    <metadata>
      <map:map xmlns:map="http://marklogic.com/xdmp/map">
        <map:entry key="validStart">
   <map:value>2019-02-06T08:00:00<map:value>
 </map:entry>
 <map:entry key="validEnd">
   <map:value>9999-12-31T11:59:59Z</map:value>
 </map:entry>
      </map:map>
    </metadata>

return  temporal:document-insert("/perso-verluste",
                                 "duesel_alex_270774.xml",
                                 $root, $options)
```

Nach der Aktualisierung enthält unsere Datenbank logisch gesehen drei Dokumente zu diesem Vorgang, die über eine Query gesucht werden können:

1. Das Originaldokument, es ist vom 1.2.2019 bis zum 5.2.2019 im System aktiv
2. Die Aktualisierung, sie ist ab dem 6.2.2019 aktiv
3. Ein "Split"-Dokument, das aus der verspäteten Aktualisierung resultiert. Es ist ab dem 6.2.2019 im System aktiv, und zeigt den Zeitraum über einen Tag, vom 4.2.2019 bis 5.2.2019 - in dem ich ohne Perso registriert war, ihn aber tatsächlich schon wieder hatte.

Im Gegensatz zu einer herkömmlichen Datenhaltung, bei der ein Datensatz aktualisiert wird - ggf. noch eine neue Version angelegt wird - wird beim Dokument-basierten Ansatz mit bi-temporaler Datenhaltung jede Transaktion separat abgespeichert.

Das ist vergleichbar mit einer Simulation des tatsächlichen Papierverkehrs bei buchhalterischen Tätigkeiten.

Die Abfrage so einer Datenbank ist dadurch nicht einfacher. Drei Queries, die jeweils eines dieser drei Dokumente zurückgeben, könnten bspw. so aussehen:

Rückgabe des Originals

```
xquery version "1.0-ml";cts:search(fn:doc(), cts:period-range-query(
  "system",
  "ISO_CONTAINS",
  cts:period(xs:dateTime("2019-02-02T00:00:00"),
             xs:dateTime("2019-02-03T23:59:59")) )
```

Hier wird geprüft, ob ein Dokument im System aktiv war, dass den Zeitraum vom 2.3. bis zum 3.3. umfasste (`ISO_CONTAINS`). Diese Query ist erfolgreich und gibt das Original-Dokument des Vorgangs zurück: In diesem Zeitraum war ich also mit gestohlenem Perso registriert.

Rückgabe des Split-Dokuments

```
xquery version "1.0-ml";cts:search(fn:doc(), cts:period-range-query(
   "valid",
   "ALN_FINISHES",
   cts:period(xs:dateTime("2019-02-06T08:00:00"),
              xs:dateTime("2019-02-06T08:00:00")) ))
```

Bei dieser Query wird geprüft, ob es ein Dokument gibt, dass zu einem bestimmten Datum auf inaktiv gesetzt wurde (`ALN_FINISHES`) - Das Split-Dokument wird automatisch auf inaktiv gesetzt, wenn die neue Version angelegt wird. Unser Suchdatum wäre also folgendes `2019-02-06T08:00:00` .

Rückgabe von Split und neuer Version

```
xquery version "1.0-ml";cts:search(fn:doc(), cts:period-range-query(
   "system",
   "ALN_AFTER",
   cts:period(xs:dateTime("2019-02-05T11:00:00"),
              xs:dateTime("2019-02-05T11:20:00")) ))
```

Hier wird geprüft ob es Dokumente gibt, die nach einer bestimmten Zeitspanne im System aktiv waren. Man beachte hier, dass eine Periode angegeben ist, obwohl nur ein Datum notwendig wäre. Der Vergleichsoperator hierzu heisst `ALN_AFTER` .

Rückgabe von aktueller Version

```
xquery version "1.0-ml";cts:search(fn:doc(), cts:period-compare-query(
   "system",
   "ISO_CONTAINS",
   "valid" ))
```

Die aktuelle Version kann in Erfahrung gebracht werden, indem geprüft wird, welche gültigen Dokumente innerhalb der Systemzeitspanne liegen. Das kann nur die aktuelle Version sein. Frühere gültige Versionen und Split-Dokumente wären vor der Systemzeit-Spanne.

Die letzte Version eines Dokuments kann aber auch einfach über ein `latest` flag in Erfahrung gebracht werden:

```
xquery version "1.0-ml";cts:search(fn:doc(), cts:and-query((
    cts:collection-query(("koolorder.xml")),
    cts:collection-query(("latest")))))
```

Weiterführende Links

- Temporal Developer's Guide auf den MarkLogic Doku-Seiten[107)]
- A Deep Dive into Bitemporal[108)]

107) https://docs.marklogic.com/guide/temporal
108) https://www.marklogic.com/blog/bitemporal/

- Temporale Datenhaltung in der Praxis mit Java[109)]

3.3.2.1 Anlegen des Testszenarios auf der ML Konsole

Die Codefragmente aus dem vorherigen Kapitel sind folgend für eine ML Konsolensitzung aufbereitet:

- Anlegen der temporalen Properties: `validStart` , `validEnd` , `systemStart` , `systemEnd`
- Anlegen der Indizes zum Suchen über Zeitbereiche: `database-range-field-index("dateTime", "systemStart",...`
- Anlegen der zwei Zeitachsen `system` und `valid`
- Anlegen der temporalen Collection `/perso-verluste`
- Anlegen des Originals am 1.2.2019
- Aktualisierung am 6.2.2019

```
xquery version "1.0-ml";

import module namespace admin =
  "http://marklogic.com/xdmp/admin" at "/MarkLogic/admin.xqy";
import module namespace temporal =
  "http://marklogic.com/xdmp/temporal" at "/MarkLogic/temporal.xqy";

declare namespace local = 'local:';
declare variable $db := "alex-test";

declare function local:create-temporal-fields()
{
  let $config := admin:get-configuration(),
      $dbid := xdmp:database($db)
  return
    try {
      admin:save-configuration(
        admin:database-add-field($config, $dbid,
        admin:database-metadata-field("validStart"))),
      admin:save-configuration(
        admin:database-add-field($config, $dbid,
        admin:database-metadata-field("validEnd"))),
      admin:save-configuration(
        admin:database-add-field($config, $dbid,
        admin:database-metadata-field("systemStart"))),
      admin:save-configuration(
        admin:database-add-field($config, $dbid,
        admin:database-metadata-field("systemEnd")))
    } catch ($err) {}
};

declare function local:create-range-index-fields()
{
  let $config := admin:get-configuration(),
      $dbid := xdmp:database($db)
  return
    try {
      admin:save-configuration(
        admin:database-add-range-field-index($config, $dbid,
        admin:database-range-field-index("dateTime", "validStart", "", fn:true()))),
      admin:save-configuration(
```

109) https://www.heise.de/developer/artikel/Temporale-Datenhaltung-in-der-Praxis-mit-Java-2100268.html?seite=all

```
        admin:database-add-range-field-index($config, $dbid,
        admin:database-range-field-index("dateTime", "validEnd", "", fn:true()))),
      admin:save-configuration(
        admin:database-add-range-field-index($config, $dbid,
        admin:database-range-field-index("dateTime", "systemStart", "", fn:true()))),
      admin:save-configuration(
        admin:database-add-range-field-index($config, $dbid,
        admin:database-range-field-index("dateTime", "systemEnd", "", fn:true())))
    } catch ($err) {}
};

declare function local:create-axes()
{
  try {
    let $t1 := temporal:axis-create(
          "valid",
          cts:field-reference("validStart", "type=dateTime"),
          cts:field-reference("validEnd", "type=dateTime")),
        $t2 := temporal:axis-create(
          "system",
          cts:field-reference("systemStart", "type=dateTime"),
          cts:field-reference("systemEnd", "type=dateTime"))
     return ()
   } catch ($err) {}
};

declare function local:create-temporal-collection()
{
  try {
    let $t:= temporal:collection-create("/perso-verluste", "system", "valid")
    return ()
  } catch ($err) {}
};

declare function local:insert-original()
{
  let $root :=
    <vorgang>
      <perso-id>XYZ</perso-id>
      <name>Alex Düsel</name>
      <status>gestohlen</status>
    </vorgang>,
     $options :=
    <options xmlns="xdmp:document-insert">
      <metadata>
        <map:map xmlns:map="http://marklogic.com/xdmp/map">
          <map:entry key="validStart">
           <map:value>2019-02-01T08:23:11</map:value>
         </map:entry>
         <map:entry key="validEnd">
           <map:value>9999-12-31T11:59:59Z</map:value>
         </map:entry>
         </map:map>
      </metadata>
    </options>
  return  temporal:document-insert("/perso-verluste",
                                   "duesel_alex_270774.xml",
                                   $root, $options)
};

declare function local:insert-update()
{
  let $root :=
    <vorgang>
      <perso-id>XYZ</perso-id>
      <name>Alex Düsel</name>
      <status>gefunden</status>
```

```
    </vorgang>,
    $options :=
    <options xmlns="xdmp:document-insert">
      <metadata>
        <map:map xmlns:map="http://marklogic.com/xdmp/map">
          <map:entry key="validStart">
           <map:value>2019-02-06T08:00:00</map:value>
          </map:entry>
          <map:entry key="validEnd">
            <map:value>9999-12-31T11:59:59Z</map:value>
          </map:entry>
          </map:map>
      </metadata>
    </options>
  return  temporal:document-insert("/perso-verluste",
                                   "duesel_alex_270774.xml",
                                   $root, $options)
};

( xdmp:invoke-function(local:create-temporal-fields#0),
  xdmp:invoke-function(local:create-range-index-fields#0),
  xdmp:invoke-function(local:create-axes#0),
  xdmp:invoke-function(local:create-temporal-collection#0),
  xdmp:invoke-function(local:insert-original#0),
xdmp:sleep(50000),
  xdmp:invoke-function(local:insert-update#0))
```

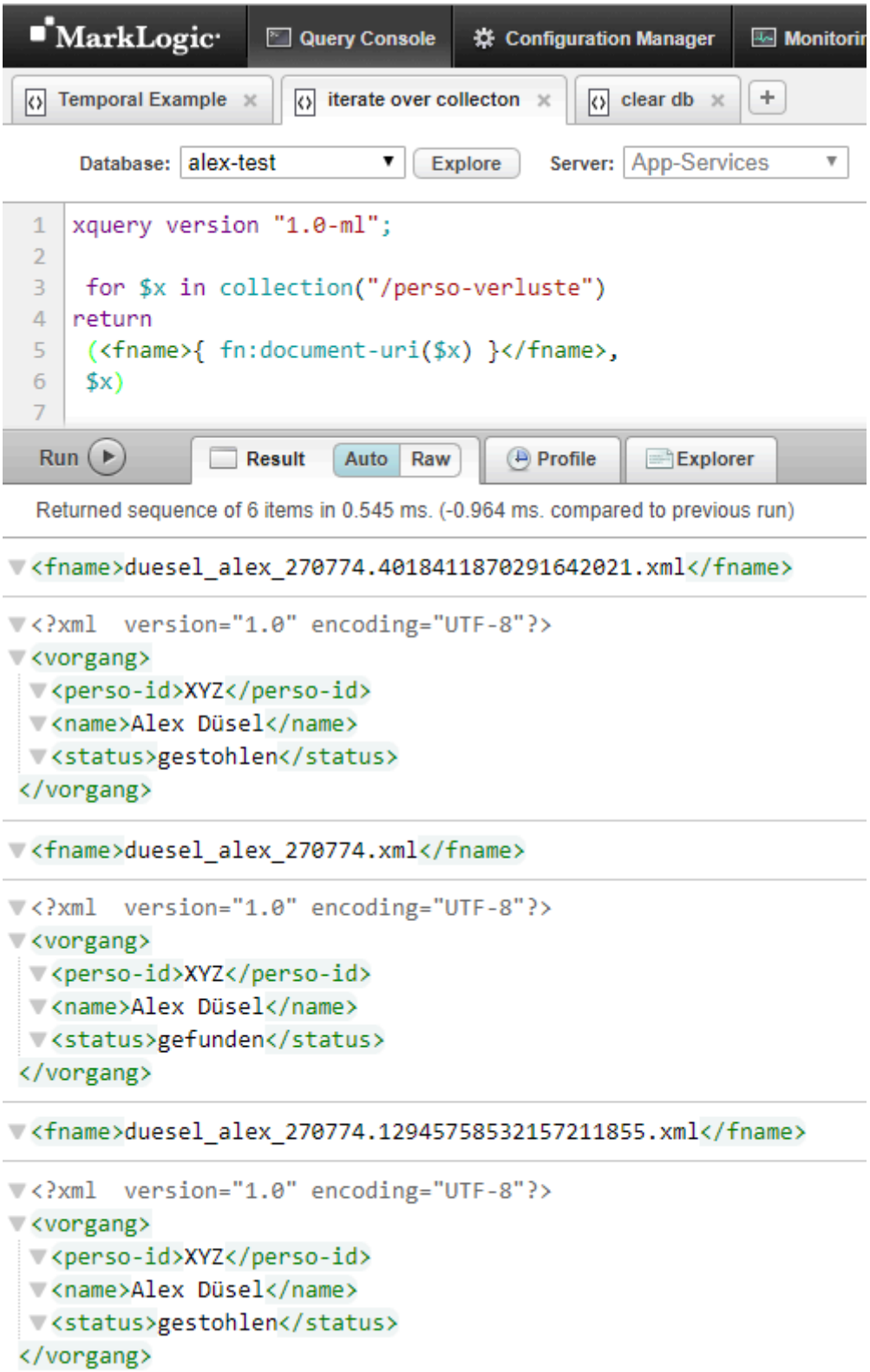
▶ Beachtenswert ist hier,

1. dass die einzelnen Schritte als Funktion über `xdmp:invoke-function` aufgerufen werden. Dieses Konstrukt wird normalerweise benutzt um eine Funktion anonym[110)] zu deklarieren und als Transaktion aufzurufen. Marklogic bietet weitere Möglichkeiten[111)] transaktional zu arbeiten.
2. Um die 5 Tage zwischen Verlustmeldung und Wiederauffinden zu simulieren, wurde zwischem dem Anlegen der Dokumente ein `xdmp:sleep` Statement eingefügt.

Lassen wir diese Query auf einer frischen Datenbank laufen, so erhalten wir die folgendes Ergebnis:

110) https://de.wikipedia.org/wiki/Anonyme_Funktion
111) https://docs.marklogic.com/guide/app-dev/transactions

Nach der Ausführung obiger Query gibt es in der DB drei Dokumente, das Orginal, das Split-Dokument und die Aktualisierung. Das Split-Dokuments und das Original sind als Vorgänger mit Suffix gekennzeichnet.



3.3.2.2 Ausführen einiger Beispiel-Queries

Wie im vorherigen Kapitel beschrieben, muss in der Konsolensitzung zwischen Anlegen der Originalversion und der Aktualisierung eine Verzögerung eingebaut werden, um irgendwie das "Zeitloch" zu simulieren, in dem - wenn wir bei dem vorherigen Beispiel bleiben - der Personalausweis als Verlust gemeldet war, aber tatsächlich schon wieder in meinem Besitz war.

Ohne jetzt groß *validStart* und *validEnd* anzupassen, habe ich das Beispiel mit einer Verzögerung von 50 Sekunden lassen:

```
( xdmp:invoke-function(local:create-temporal-fields#0),
  xdmp:invoke-function(local:create-range-index-fields#0),
  xdmp:invoke-function(local:create-axes#0),
  xdmp:invoke-function(local:create-temporal-collection#0),
  xdmp:invoke-function(local:insert-original#0),
  xdmp:sleep(50000),
  xdmp:invoke-function(local:insert-update#0))
```

Und bekomme folgendes Ergebnis:

Diese Daten wurde aus der Exploreransicht der Konsolensitzung entnommen.

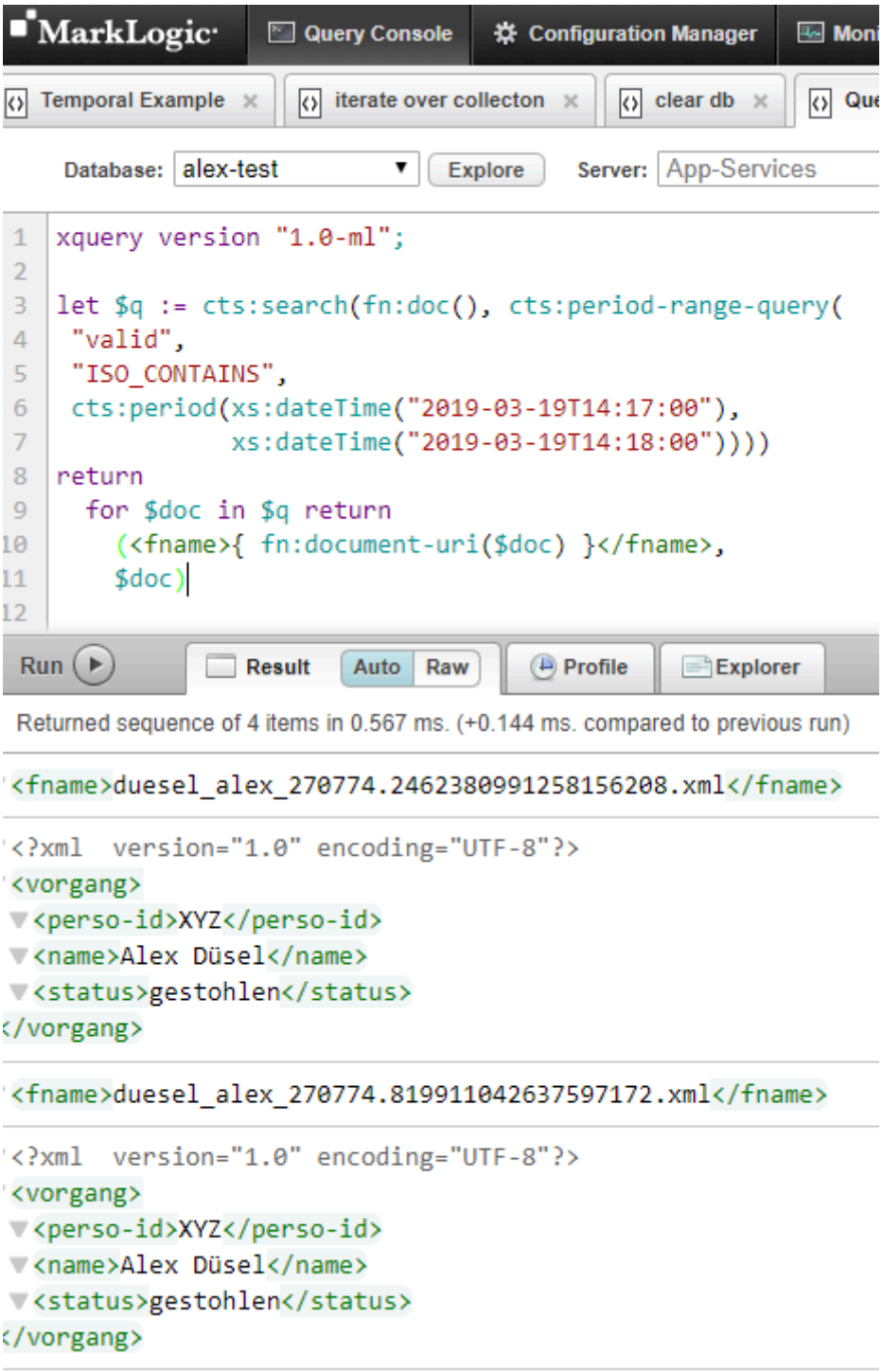
Document	validStart	validEnd	systemStart	systemEnd
due-sel_alex_270774.24	2019-03-19T14:17:00	**unendlich**	2019-03-19T13:18:28	2019-03-19T13:19:18
due-sel_alex_270774.81	2019-03-19T14:17:00	2019-03-19T14:18:00	2019-03-19T13:19:18	**unendlich**
due-sel_alex_270774.xr	2019-03-19T14:18:00	**unendlich**	2019-03-19T13:19:18	**unendlich**

Wie man sieht, müsste sich jetzt das Zeitloch von 50 Sekunden der Systemzeiten innerhalb der gültigen Zeiten befinden, um das Beispiel mit dem verlorenen Ausweis zumindest für die Zeitspanne von 50 Sekunden simulieren zu können.

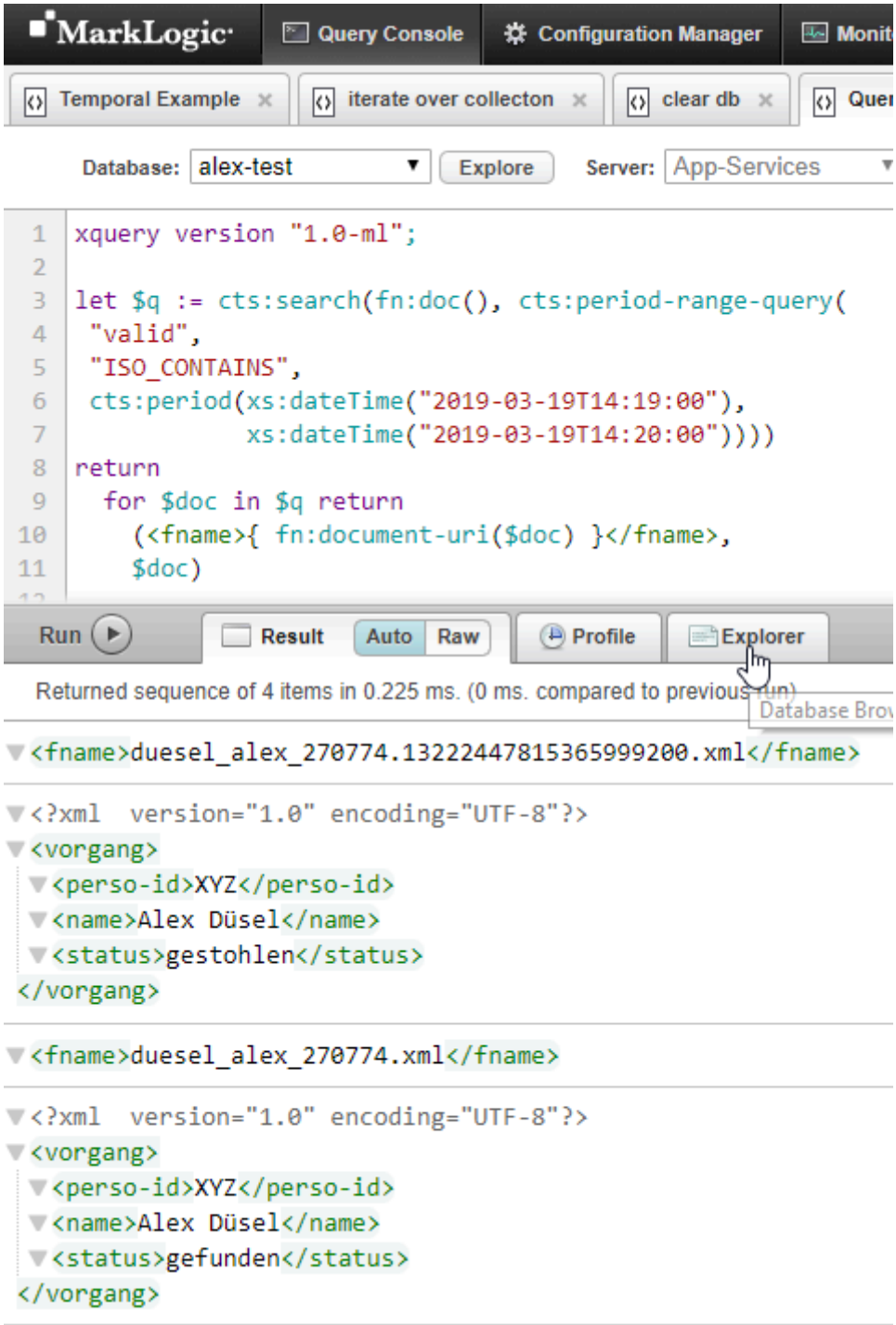
Das korrekte Setup bleibt an dieser Stelle dem geneigten Leser selbst überlassen.

Zwei Queries, die die Werte in der Tabelle illustrieren, sind bspw. folgende:

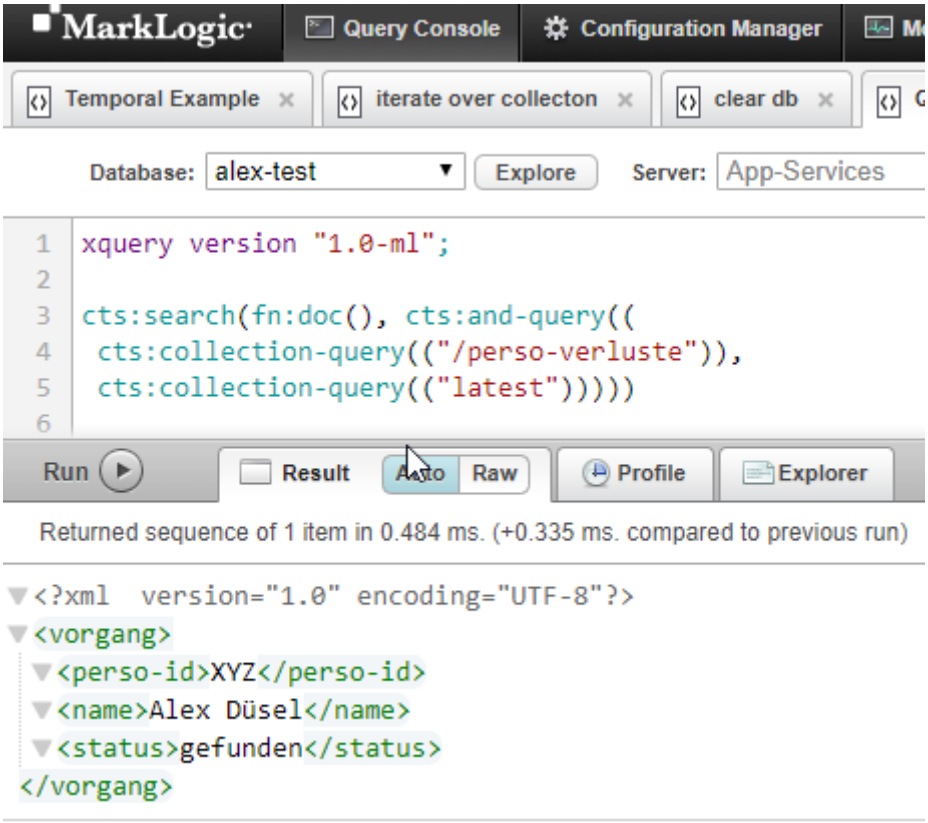
Bei dieser Query werden alle Dokumente gesucht, deren gültiger Zeitraum, den Zeitraum zwischen 14:17 Uhr und 14:18 Uhr umfasst. Das sind das Split-Dokument und das Original-Dokument. Beide haben den Status "gestolen", da erst um 14:18 Uhr der Fund bekannt gegeben wurde.



Verändert man den Zeitraum nach 18:00 Uhr, so wird das Original - wie erwartet - durch das Update verdrängt.



Die aktuelle Version des temporal verwalteten Dokuments ist in der Collection "latest" gespeichert.



Wie man sieht, kann man - ausreichend Lösungsphantasie vorausgesetzt - einiges mit diesen Queries anstellen. Hilfreich ist sicherlich auch die umfangreiche Liste der Vergleichsoperatoren, die MarkLogic zum Vergleichen von Zeiträumen bereitstellt: ISO Operators[112] und ALLEN Operators[113].

3.3.3 Webapps mit MarkLogic

Konfiguration mit cURL

cURL[114] ist ein gebräuchliches Kommandozeilentool, mit dem man Web-Requests an einen Server schicken kann.

> ▶ Die Beispiel-Queries auf diesen Seiten wurden größtenteils von den MarkLogic Doku-Seiten übernommen, sind jedoch für Windows Rechner angepasst. Statt Shell Skripten mit diversen Besonderheiten, kann man die Code-Schnippsel auch in Batch-Dateien packen und ausführen.

MarkLogic horcht auf Port *8002* mit seiner Configuration Manager Applikation. Über diesen Port können auch cURL Requests zur Remote-Konfiguration abgesetzt werden.

112) https://docs.marklogic.com/guide/temporal/searching#id_92200
113) https://docs.marklogic.com/guide/temporal/searching#id_98704
114) https://de.wikipedia.org/wiki/CURL

Zum Anlegen einer Datenbank setzt man den folgenden cURL Befehl ab:

```
curl -X POST  --anyauth -u admin:admin --header "Content-Type:application/json"
              -d '{\"database-name\":\"xml-scrapper-content\"}'
             http://localhost:8002/manage/v2/databases
```

Auf meiner Windows Maschine konnte ich den Befehl, wie in der MarkLogic Doku[115] beschrieben, nicht ausführen, da erst das JSON mittels Backslashes maskiert werden musste. Ausserdem ist in der Powershell der ***curl*** Befehl per alias auf ein Windows Programm gemappt[116]

- **Abhilfe:** Maskieren des JSON Strings und Entfernen des cURL Aliases auf Windows.

Analog legt man einen "Forrest" an, den die oben definierte Datenbank braucht:

```
curl --anyauth --user alex:anoma66 -X POST -d '{\"forest-name\": \"xml-scrapper-forrest\",
                                                \"database\": \"xml-scrapper-content\"}'
               -i -H "Content-type: application/json" http://localhost:8002/manage/v2/forests
```

Die Konsole quittiert das erfolgreiche Ereignis mit diesen Meldungen:

```
HTTP/1.1 201 Created
Location: /manage/v2/forests/12099403305847426321
Content-type: application/xml; charset=UTF-8
Cache-Control: no-cache
Expires: -1
Server: MarkLogic
Content-Length: 0
Connection: Keep-Alive
Keep-Alive: timeout=5
```

Die Erfolgsmeldung kann man auch leicht in der Übersicht des Configuration Managers auf Port 8003 nachprüfen. Nun können wir die neue Datenbank mit der **MarkLogic Content Pump** befüllen.

Dazu laden wir folgendes Beispiel-XML, das sich in einem Ordner *input-files* befindet, in die Datenbank:

```
<test>
    <title>Test Datei</title>
    <chapter>
        <title>Test Kapitel 1</title>
```

115) https://docs.marklogic.com/REST/POST/manage/v2/databases
116) https://daniel.haxx.se/blog/2016/08/19/removing-the-powershell-curl-alias/

```
        <content>Kapitel Inhalt 1</content>
    </chapter>
    <chapter>
        <title>Test Kapitel 2</title>
        <content>Kapitel Inhalt 2</content>
    </chapter>
    <chapter>
        <title>Test Kapitel 2</title>
        <content>Kapitel Inhalt 2</content>
    </chapter>
</test>
```

▻ Den Tippfehler im obigen XML werden wir im folgenden Kapitel korrigieren.

Das geschieht mit dem Befehl:

```
mlcp import -database xml-scrapper -host localhost -username admin -password admin
            -input_file_path input-files -input_file_type aggregates
            -aggregate_record_element chapter
            -output_collections /chapter
            -output_uri_prefix /chapter/
            -output_uri_suffix .xml
```

Nun brauchen wir nun noch einen Application Server in MarkLogic anlegen, um eigene XQuery Skripte laufen lassen zu können. In einer Datei *`server-setup.json`* definieren wir unsere Server Einstellungen:

```
{ "server-name":"xml-scrapper",
  "root":"c:\\xquery",
  "port":"8088",
  "content-database":"xml-scrapper-content",
  "server-type":"http",
  "group-name":"Default"
}
```

Diese schicken wir mit dem folgenden cURL Befehl an den Server:

```
curl -X POST --digest -u alex:anoma66 -H "Content-type: application/json"
     -d @server-setup.json http://localhost:8002/manage/v2/servers
```

Im Web-Interface können wir uns überzeugen, dass alles geklappt hat:

Im Reiter `Configure` können wir den App Server auf MarkLogic konfigurieren.

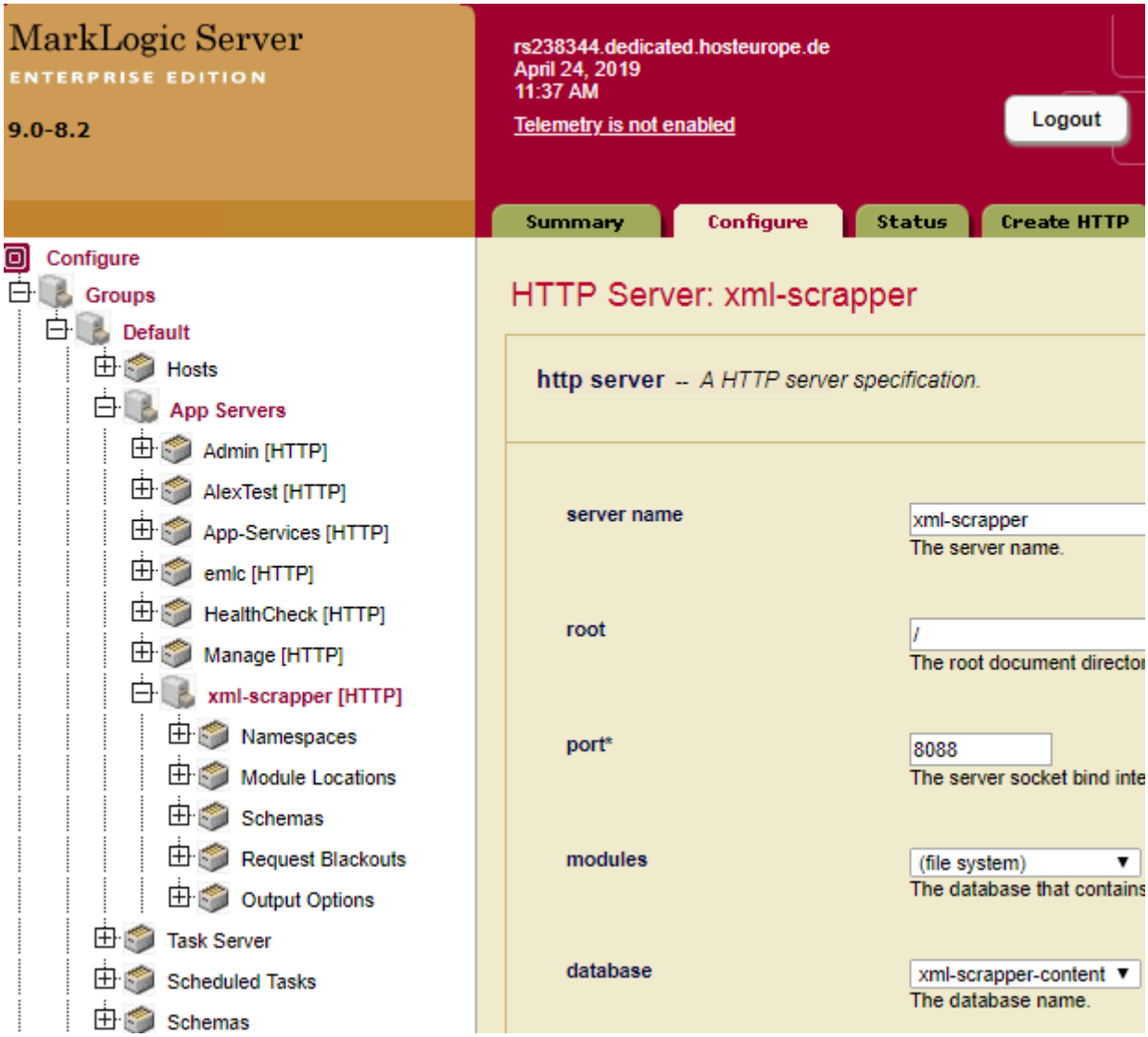


figure 12: Konfiguration eine App Servers auf MarkLogic

Die Konfiguration kann natürlich auch mittels des Webinterfaces auf Port 8001 gemacht werden, oder aber auch per XQuery Skripte über die Konsole auf Port 8000. Einige Skripte für diesen Zweck befinden sich auf den Developer Seiten von MarkLogic[117)]

Nach diesen Vorbereitungen können wir unseren App-Server nun mit Skripten bestücken, wie im folgenden Abschnitt beschrieben. Wir legen die Skripte in das Wurzelverzeichnis `c:\xquery` , das wir oben definiert haben und können diese über einen Webrequest aufrufen, z.B. so:

```
http://localhost:8088/test.xqy
```

117) https://docs.marklogic.com/guide/admin-api/configure

Implementierung als XQuery Skript

In diesem Abschnitt werden wir eine HTML Seite mit Inhaltsverzeichnis aus den zuvor geladenen Daten generieren.

Beginnen wir mit einem Skript `book.xqy` im Verzeichnis `C:\xml-scrapper`

```
xquery version "1.0-ml";
xdmp:set-response-content-type("text/html"),
let $pages :=
<html>
    <body>
        {
            for $chapterin collection("/chapter")/descendant::chapter
                return (
                    <h3>{ $chapter/title/text() }</h3>,
                    <p>{ $chapter/content/text() }</p>
                )
        }
    </body>
</html>
return $pages
```

Ergebnis: Die Kapitel der Webseite werden hintereinander weggeschrieben. Das ist natürlich noch nicht optimal

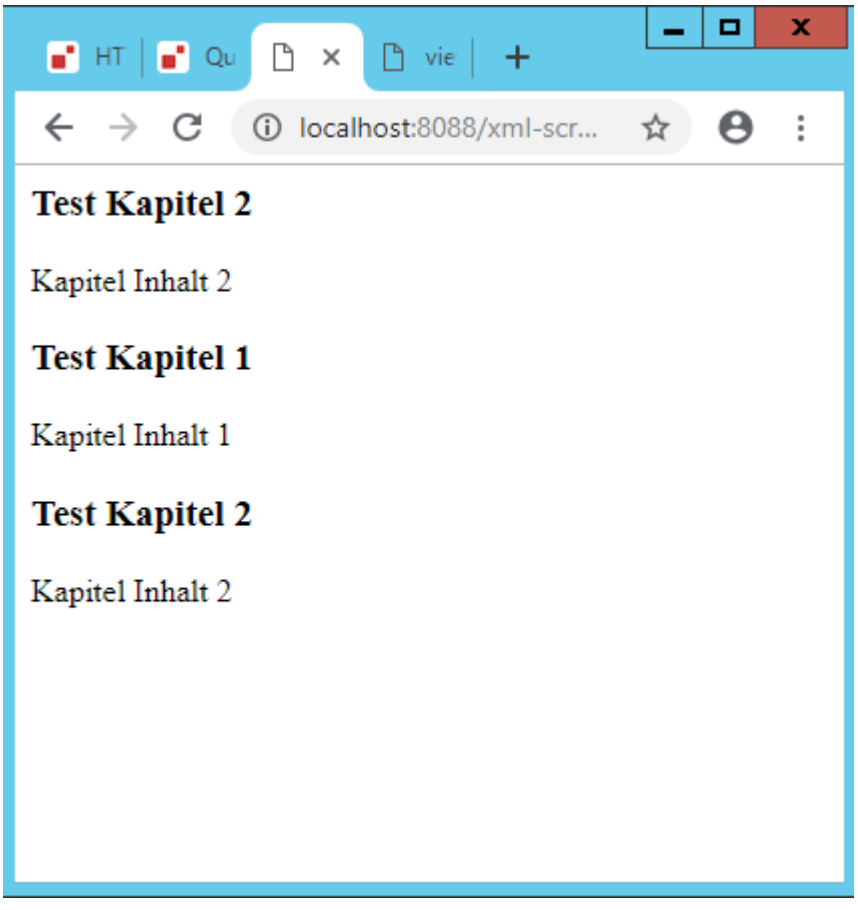


figure 13: Erste Ausgabe unseres kleinen XQuery Skripts für eine Website

Hier fällt auf, dass wir 2x ein Kapitel 2 eingebunden haben. Es handelt sich dabei um einen Tippfehler.

⊳ Wir kümmern uns um diesen Fehler später.

Nun wollen wir die einzelnen Seiten auf verschiedene Webseiten aufsplitten und auf einer Cover-Page ein Inhaltsverzeichnis darstellen.

```
xquery version "1.0-ml";

declare variable $page:= xdmp:get-request-field('page');
xdmp:set-response-content-type("text/html"),
let $page-id := if ($page) then $page else ('cover'),
$pages :=
<html>
    <body>
        {
            <h3>Welcome to The Book</h3>,
            for $chapter at $positionin collection("/chapter")/descendant::chapter
                return (
                    if ($page-id = 'cover') then (
                        <p><a href="?page={$position}">{ $chapter/title/text() }</a></p>
                    ) else (
                        (: TODO :)
                    )
                )
        }
    </body>
</html>
return $pages
```

Im Vergleich zu einer XSLT Lösung stellt man fest, dass man vergeblich versucht die XPath Funktion `fn:position()` anzuwenden. Stattdessen verwendet man das Schlüsselwort `at` in der `for` Loop.

Auf der initialen Cover-Seite wird nun ein verlinktes Inhaltsverzeichnis angezeigt:

Der zweite Schritt unserer Webapplikation ist ein Inhaltsverzeichnis mit verlinkten Kapiteln

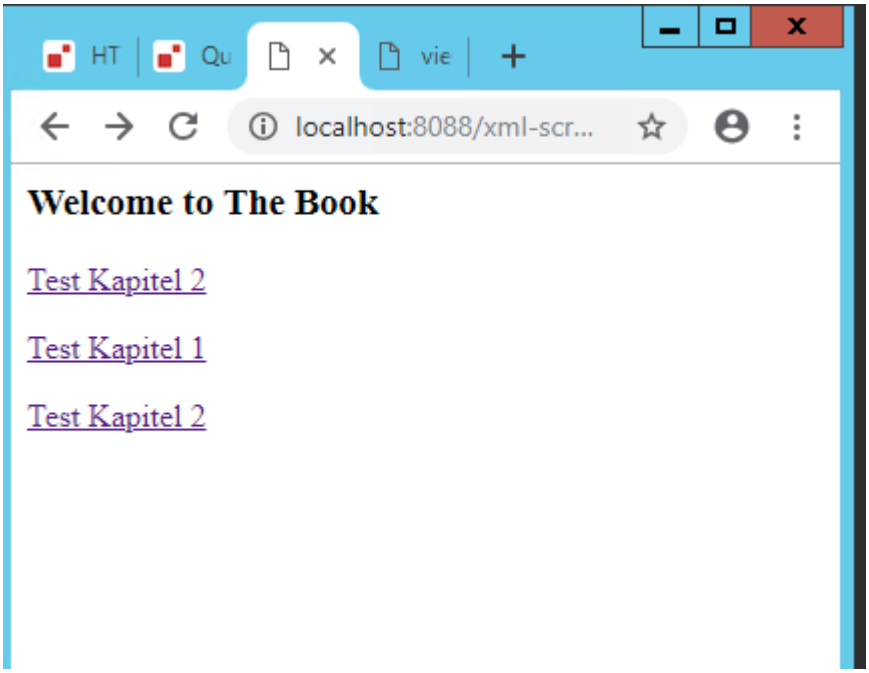


figure 14: Zweite Ausgabe unseres kleinen XQuery Skripts für eine Website

Der im Skript deklarierte Request-Parameter `$page` wird nun ausgewertet, um die Kapitelseiten zu erzeugen.

```
xquery version "1.0-ml";

declare variable $page:= xdmp:get-request-field('page');
xdmp:set-response-content-type("text/html"),
let $page-id := xs:decimal(if ($page) then $page else '0'),
$pages := collection("/chapter"),
$website :=
<html>
    <body>
        {
            <h3>Welcome to The Book</h3>,
            for $chapter at $positionin $pages/descendant::chapter
                return (
                    if ($page-id lt 1 or  $page-id gt count($pages)) then (
                        <p><a href="?page={$position}">{ $chapter/title/text() }</a></p>
                    ) else  if ($page-id = $position) then (
                        <h2>{ $chapter/title/text() }</h2>,
                        <p>{ $chapter/content/text() }</p>,
                        <p><a href="{ xdmp:get-request-path() }">Back To Cover</a></p>
                    ) else ()
                )
        }
    </body>
</html>
return $website
```

Hier ist das `at` Schlüsselwort interessant mit dem man die Position in der Schleife abgreifen kann. `fn:position()` wie bei XSLT gebräuchlich würde hier nicht

funtktionieren. Dass wir bedingte Anweisungen in funktionalen Sprachen als Ausdruck auswerten können, haben wir in hier schon gelernt, vgl. die `sx:decimal` Cast Anweisung zur Typ-Konvertierung.

Unsere Website wäre eigentlich schon perfekt, wenn da der fehlerhafte Datenimport nicht wäre, und wir das Kapitel 2 nicht doppelt importiert hätten. Um die Daten zu bereinigen ist eine Daten-Migration notwendig.

Webapps mit mehreren Datenbanken

Jeder App-Server ist in MarkLogic genau einer Content-Database zugeordnet. Darin sollten alle Daten vorhanden sein, auf die die Webapplikation zugreift.

Es ist jedoch über einen kleinen "Hack" möglich andere Datenbanken in derselben MarkLogic Webapp abzufragen. Dazu verwendet man die `xdmp:eval-in` Funktion.

Wie der Name schon sagt, wird damit ein Ausdruck *in* einer anderen Datenbank *evaluiert*.

Das Beispiel dazu auf der MarkLogic-Doku Seite[118)] sieht folgendermassen aus:

```
xquery version "0.9-ml"
declare namespace my='http://mycompany.com/test'

let $s :=
      "xquery version '0.9-ml'
       declare namespace my='http://mycompany.com/test'
       define variable $my:x as xs:string external
       concat('hello ', $my:x)"
return
    (: evaluate the query string $s using the variables
       supplied as the second parameter to xdmp:eval :)
    xdmp:eval-in($s,
                 xdmp:database("Documents"),
   ("my:x"),
   "world"))

=> hello world
```

Ein Anwendungsbeispiel aus der Praxis würde demnach so aussehen:

```
declare function local:remove-sql-view($sql-view-name) {
  let $url := concat('/sql-views/', $sql-view-name)
  return xdmp:eval-in('xquery version "1.0-ml";
              declare variable $url as xs:string external;
              xdmp:document-delete($url),
              xdmp:schema-database(),
              ('url'),$url)
  )
};
```

Hier wird eine zuvor gesetzte SQL View, vgl. Kapitel SQL Views in MarkLogic on page 96, die per Default in der Schema-Datenbank untergebracht ist, wieder aus dem System gelöscht.

118) https://docs.marklogic.com/xdmp:eval-in

Datenkorrektur mit der Konsole

Um die fehlerhaften Daten aus dem vorherigen Kapitel zu korrigieren, öffnen wir eine Konsolensitzung auf Port `8000` :

▻ Wir bemerken in der folgenden Abbildung, dass das `mlcp` Kommando den `document-name` mit dem absoluten Pfad der Datei im Dateisystem des importierenden Rechners geprefixt hat.

Auf der Konsole können wir uns die in der Collection abgespeicherten Dokumente auflisten lassen.

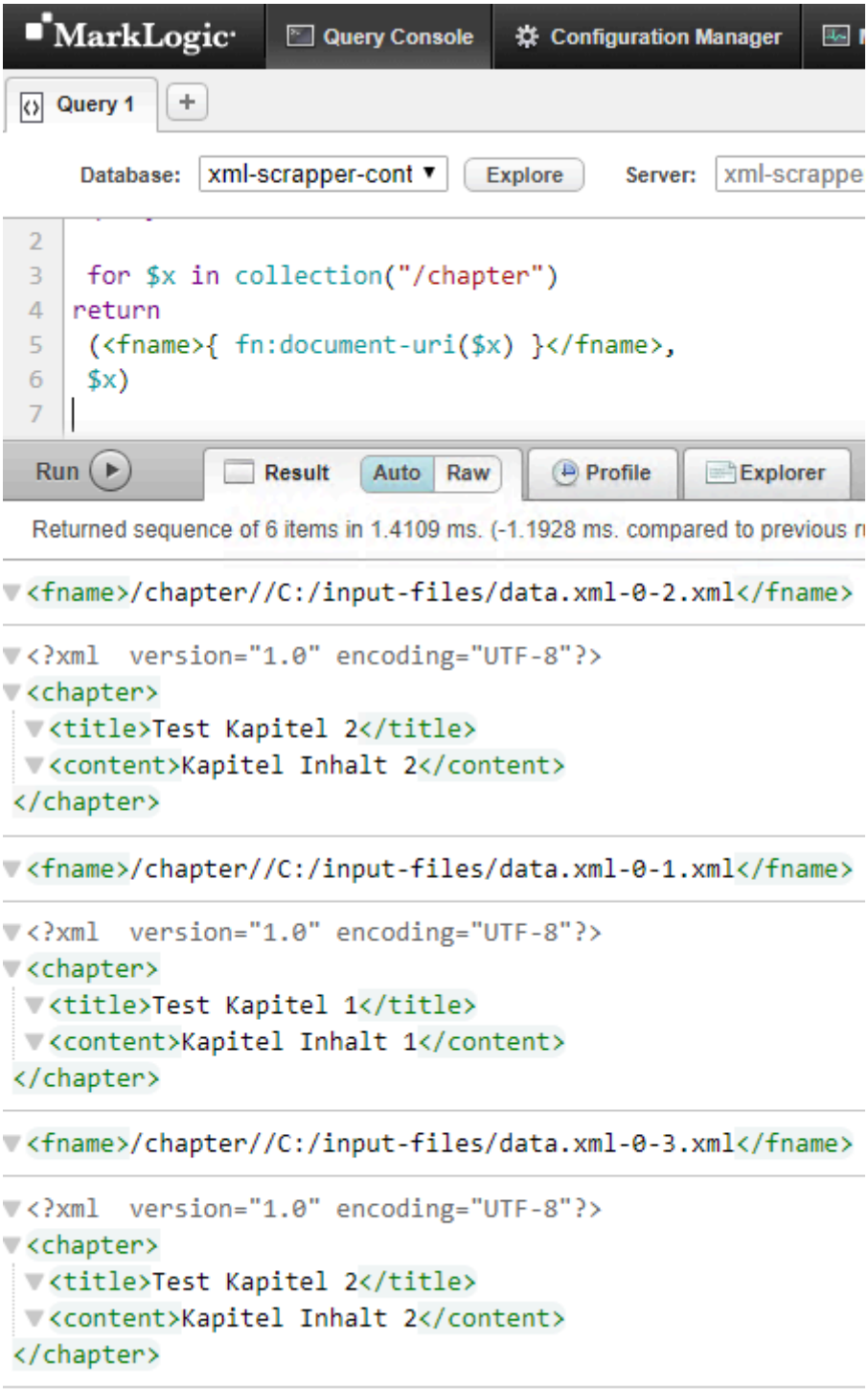


figure 15: MarkLogic Konsolensitzung mit einer Collection Iteration

Wir sehen, dass wir zweimal ein Kapitel 2 in der Collection angelegt haben. Wir müssen also das 3. Element in der Collection korrigieren:

```
xdmp:node-replace(doc("/chapter//C:/input-files/data.xml-0-3.xml")/chapter/title,
    <title>Test Kapitel 3</title>);xdmp:node-replace(doc("/chapter//C:/input-files/data.xml-0-3.xml")/c

    <title>Kapitel Inhalt 3</title>);
```

Das Semikolon zum Abschluss des Statements ist eine Besonderheit von MarkLogic und gibt an, dass dieses Statement in einer Transaktion ausgeführt werden soll.

- In anderen XQuery Implementierungen gibt es diese Funktion möglicherweise nicht.

Nach dieser Korrektur sollten die Daten wieder stimmen und unsere Webapp ist fertig...

3.3.3.1 Wikipedia Scrapper Applikation

In diesem Kapitel wollen wir eine XSLT Transformation bauen, die während des Durchlaufs durch einen XML Baum Anfragen an ein XQuery Skript auf einem MarkLogic Server stellt. Über einen GET Request wird ein Feld `<title>` zur Persistierung in einer ML Collection übertragen.

App Server Authentifizierung

Um dieses Szenario realisieren zu können, müssen wir die Rechte in MarkLogic so einstellen, dass Webrequests ohne eine Authentifizierung akzeptiert werden. D.h. unsere Applikation ist also eher für den *internen* Gebrauch gedacht.

Da momentan der Saxon XSLT Prozessor die Auflösung von URIs dem Java URI Resolver überlässt[119], dieser aber eine Authentifizierung mit Credentials in der URL, wie in

```
fn:json-to-xml('http://admin:admin@localhost:8088')
```

noch nicht unterstützt, müssen wir die Authentifizierung für unsere MarkLogic Webapp ausschalten.

Dazu setzen wir die Einstellung `authentication` unseres App Servers auf `application-level` und weisen die Default-User Rolle einem `admin` Benutzer zu.

119) https://sourceforge.net/p/saxon/mailman/message/13252035/

In der App Server Konfiguration kann man die Stufe einstellen, auf der der Zugriffsmechanismus greifen soll. Hier stellen wir `application-level` mit einem Admin-User ein, um für das Intranet die Authentifizierung auszuschalten.

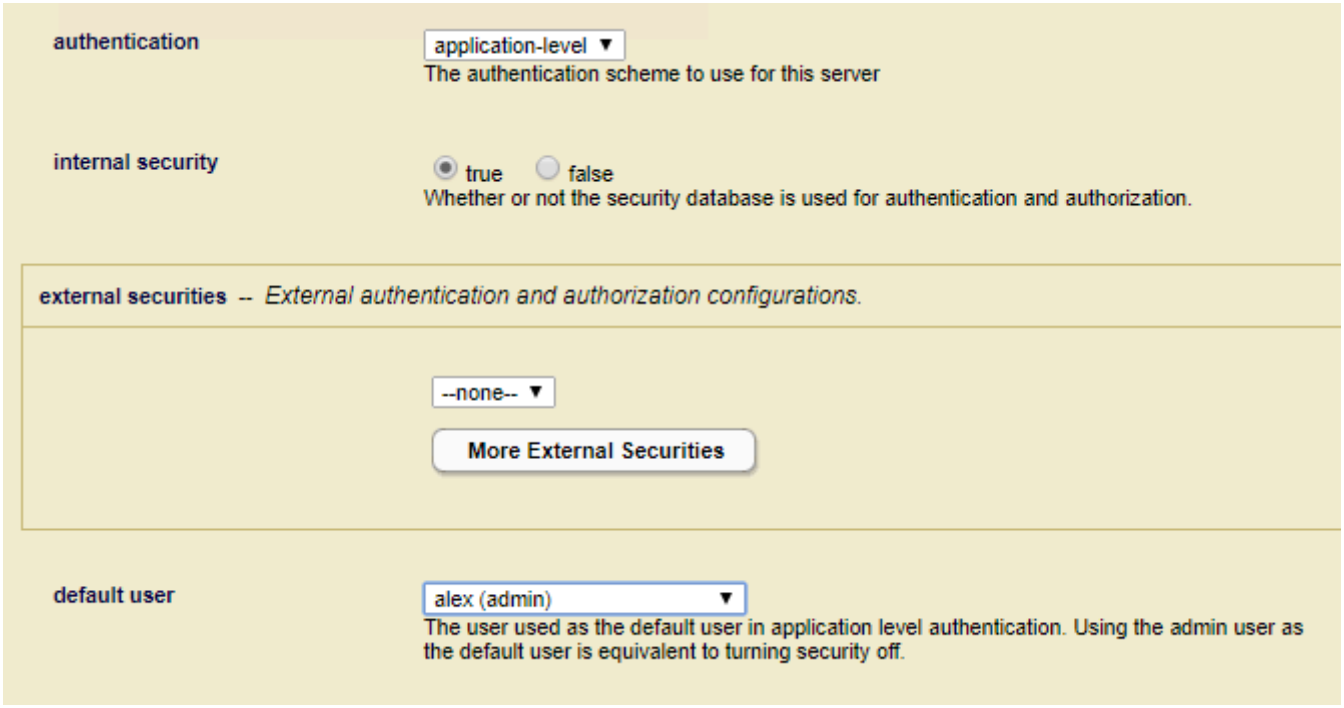


figure 16: MarkLogic App Server Authentifizierung einstellen

Eine ausgefeiltere Einstellung wird im Kapitel Dokument-Rechte in MarkLogic on-page 133 beschrieben.

XML Eingabe

Damit das Experiment etwas aufregender wird, arbeiten wir mittels XML Streaming auf einem Wikipedia Dump mit 5.3 GB Filesize[120]. Das XML dazu sieht folgendermassen aus:

```
<mediawiki xmlns="http://www.mediawiki.org/xml/export-0.10/"
             xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
             xsi:schemaLocation="http://www.mediawiki.org/xml/export-0.10/
                                 http://www.mediawiki.org/xml/export-0.10.xsd"
             version="0.10"
             xml:lang="en">

   <siteinfo>
     <sitename>Wikipedia</sitename>
     <dbname>enwiki</dbname>
     <base>https://en.wikipedia.org/wiki/Main_Page</base>
     <generator>MediaWiki 1.29.0-wmf.12</generator>
     <case>first-letter</case>
     <namespaces>
       [...]
     </namespaces>
   </siteinfo>
   <page>
     <title>AccessibleComputing</title>
     <ns>0</ns>
     <id>10</id>
```

120) https://dumps.wikimedia.org/enwiki/latest/

```
    <redirect title="Computer accessibility" />
    <revision>
      <id>631144794</id>
      <parentid>381202555</parentid>
      <timestamp>2014-10-26T04:50:23Z</timestamp>
      <contributor>
        <username>Paine Ellsworth</username>
        <id>9092818</id>
      </contributor>
[...]
```

Wir wollen alle Titel in einer Datenbank speichern, deshalb wird auf das `<title>` Element gematcht.

XSLT Transformation

Die Streaming Transformation mit dem Iterator Konzept sieht so aus:

```
template name="main">
  <xsl:source-document href="{$input-file}" streamable='yes'>
    <result>
      <xsl:iterate select="page/title">
        <xsl:variable name="json-call" select="json-to-xml(
                                               unparsed-text(
                                               concat($server-url,'/scrap-title.xqy?',
                                                      'title=',encode-for-uri(.))))"/>
        <state>
          <xsl:sequence select="$json-call/descendant::*[@key='state']/text()"/>
        </state>
      </xsl:iterate>
    </result>
  </xsl:source-document>
</xsl:template>
```

Hier werden in einer Ergebnis Struktur mit einem `<result>` Element einzelne `<state>` Elemente ausgegeben.

Der Inhalt dieser Elemente ist Rückgabewert eines Webrequests über die Funktion `fn:unparsed-text()` .

An das Skript `scrap-title.xqy` wird ein Parameter `title` übergeben:

```
xquery version "1.0-ml";

import module namespace json = "http://marklogic.com/xdmp/json"
                          at "/MarkLogic/json/json.xqy";

declare namespace local = 'local:';
declare variable $title := xdmp:get-request-field('title');

declare function local:render-response($response)
{
  xdmp:add-response-header("Pragma", "no-cache"),
  xdmp:add-response-header("Cache-Control", "no-cache"),
  xdmp:add-response-header("Expires", "0"),
  xdmp:set-response-content-type('text/json; charset=utf-8'),
  xdmp:unquote($response)
};

let $root := <title>{ $title }</title>,
$options :=
<options xmlns="xdmp:document-insert">
  <permissions>{ xdmp:default-permissions() }</permissions>
  <collections>
```

```
    <collection>/wikimedia-titles</collection>
  </collections>
</options>,
$fname := concat('/wikimedia-titles/', xdmp:md5($title), ".xml"),
$td := xdmp:document-insert($fname, $root, $options)
return
  local:render-response(concat('{"state":"success","title":"',$title,'"}'))
```

Wenn alles gut läuft, sollte die Transformation nach einer halben Stunde abgeschlossen sein. Es sollten sich in der

Collection `/wikimedia-titles` viele Einträge befinden, mit Dateinamen wie:

```
<fname>/wikimedia-titles/b00bb36cf9dd18f12141f463f59947e6.xml</fname>
```

3.3.4 Dokument-Rechte in MarkLogic

MarkLogic bietet ein sehr ausgefeiltes Rechte-und Rollensystem. Da wir MarkLogic aber vornehmlich als Datenbank einsetzen und nicht als Content Management System, reicht es aus, die grundlegende Funktionalität zu kennen.

> ⊳ Ohne weitere Massnahmen werden Dokumente mit den Rechten des Erzeugers versehen.

Lädt man z.B. Dokumente über die MarkLogic Content Pump in die Datenbank, wie mit diesem Befehl:

```
mlcp_opts="-database alex-test -host localhost -username admin -password admin"

mlcp import $mlcp_opts \
     -output_permissions xml-scrapper,read,xml-scrapper,update
     -input_file_path input-files \
     -input_file_type aggregates \
     -aggregate_record_element chapter \
     -output_collections /chapter \
     -output_uri_prefix /chapter/ \
     -output_uri_suffix .xml
```

So ist es besonders wichtig, die Einstellung:

```
-output_permissions role1,read,role2,update
```

zu setzen, wenn man z.B. für eine Webapp nur einen User mit einer bestimmten Rolle vorsieht.

Vergisst man die Option `-output_permissions` , so kann man u.U. die Dokumente von einer Webapp aus nicht zur Anzeige bringen.

- In diesem Fall sind keine Rechte an den hochgeladenen Dokumenten vorhanden. Man kann diese dann in einer Webapp nur herunterladen, wenn man in der Admin-Rolle eingeloggt ist.

Es gibt einen einfachen Weg, die Rechte an einem Dokument zu überprüfen. Dazu führt man in der Konsole die "Browse"-Aktion auf einer Datenbank aus und wechselt in den Dateireiter Permissions:

Der "Browse"-Button in der MarkLogic Konsole wird oft übersehen, bietet aber viele nützliche Funktionen, wie z.b. die Anzeige der gesetzten Dokumentrechte.

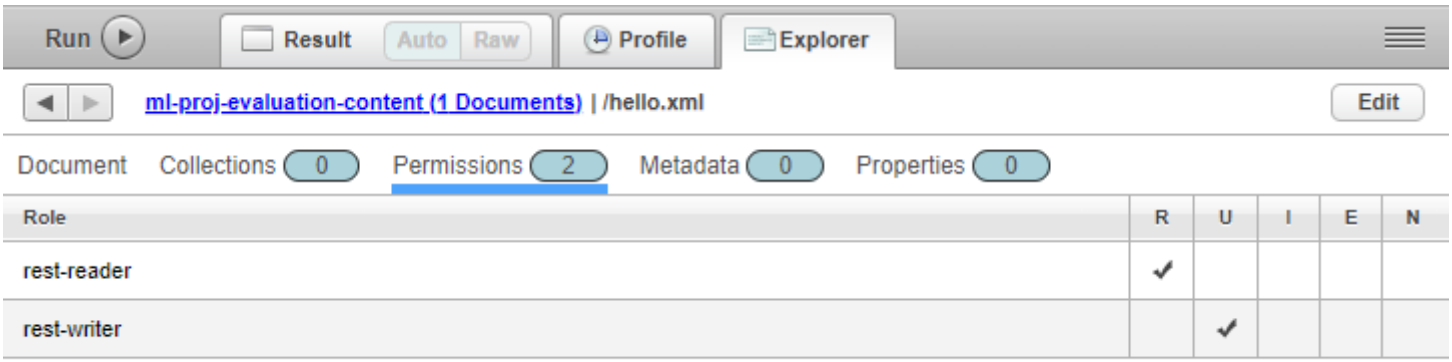


figure 17: Anzeige der Dokument-Rechte in der MarkLogic Datenbank

Ein XQuery-Skript, das eine Webapp betreibt, kann nicht nur im Browser aufgerufen werden, sondern auch von einem externen Programm bedient werden. Damit dies möglich ist, müssen die Rechte der WebApp wie in Kapitel Wikipedia Scrapper Application on page 130 gesetzt werden.

- Idealerweise sollte man dann aber auch dem User xml-scrapper eine Rolle xml-scrapper zuweisen, die nur Dokumente mit den Permissions -output_permissions xml-scrapper,read,xml-scrapper,update, vgl. oben, lesen und modifizieren kann - und nicht die Admin-Rolle.

3.3.5 MarkLogic Tools

MarkLogic bringt out-of-the-box schon eine Vielzahl an Ansichten und Tools mit.

Ein produktiver Applicatione Server sollte keine Entwicklerwerkzeuge bereitstellen, bzw. es sollte verhindert werden, dass irgendwelche halbfertigen Sachen produktiv laufen.

Deshalb ist es nicht verwunderlich, dass Ansichten in MarkLagic fehlen, wie z.B. ein Datenbrowser mit Syntaxhighlighting.

Hier springen fleissige Entwickler in die Bresche und stellen Werkzeuge zur Verfügung.

3.3.5.1 EXPath Konsole

Die MarkLogic EXPath Konsole[121)] ist eine Webanwendung, die ausserhalb des MarkLgic Servers verschiedene Ansichten auf die Daten bereitstellt. Sie umfasst auch einige praktische Tools, wie:

- Package Manager
- Browser (Für Dokumente und Tripel)
- Document Manager
- XQuery Profiler

Im Rahmen dieser Lektüre wird nur der Browser kurz vorgestellt, weil man damit unkompliziert die Daten sichten kann.

Diese werden in einem Editorfenster mit Syntaxhighlighting dargestellt.

121) https://github.com/fgeorges/expath-ml-console

Das Sichten großer Datenmengen ist mit dem EXPath Browser aus Performanzgründen nicht möglich.

⊳ Auf diesen Umstand weist der Autor explizit hin. Die Anwendung sollte nicht produktiv eingesetzt werden, sondern ist für Entwicklungszwecke gedacht.

Nach der Installation läuft die EXPath Konsole standardmässig auf Port 8010. Über den Reiter **Databases** kann man sich die definierten Databases anschauen:

Über den Reiter **Databases** gelangt man zur Datenbank-Ansicht. Hier kann man die einzelnen Verzeichnisse mittels des **Browse-Buttons** auswählen.

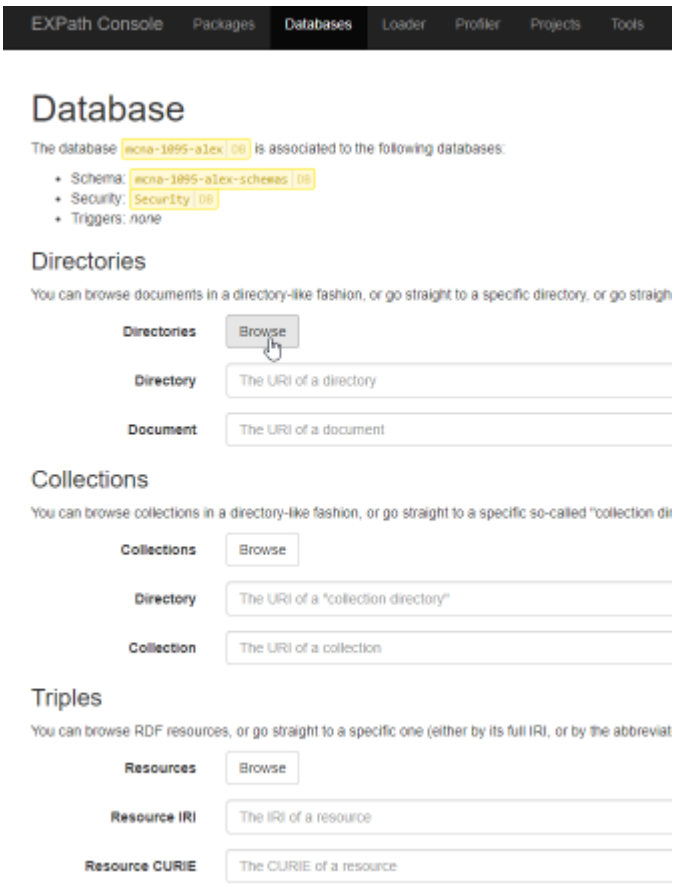


figure 18: Datenbank Ansicht im EXPath Explorer für MarkLogic

Über einen Dateipfad gelangt man zur Datenansicht mit Syntax-Highlighting.

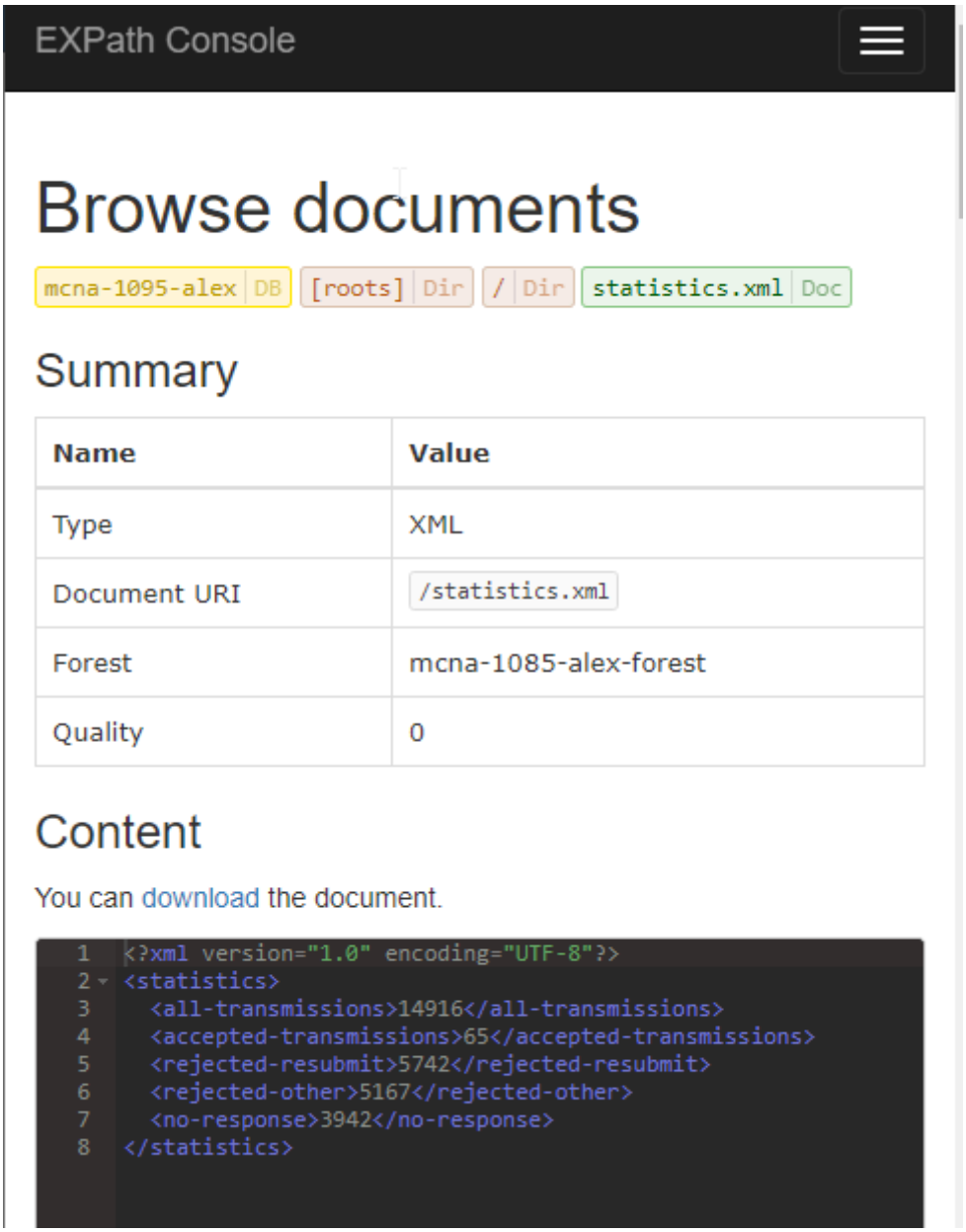


figure 19: Datensicht mit Syntax-Highlighting in der EXPath Konsole für MarkLogic

Die EXPath Konsole erwartet, dass der MarkLogic auf den Standard-Ports läuft. Das ist ***8001*** für die Admin-Applikation. Ist dies nicht der Fall, so schlägt eine Installation mittels des Tools mlproj[122] fehl.

▻ Wie dieser Umstand korrigiert werden kann, lässt sich bestimmt über die Projekt-Seite auf GitHub erfragen. Da ich aus anderen Gründen MarkLogic von einem exotischen Port wieder auf den Standardport zurückgesetzt habe, habe ich diese Option übersprungen.

3.3.5.2 mlcp - MarkLogic Content Pump

Die Content Pump für MarkLogic ist ein Java Tool, das den Bulk-Import von Daten über die Kommadozeile realisiert.

Das betreffende GitHub Projekt befindet sich hier[123].

Zur einfachen Installation kann man sich aber auch die Binaries auf den

Developer Seiten[124] herunterladen.

Folgendes Bash Skript benutze ich, um Daten nach MarkLogic hochzuladen:

```
#!/bin/bash

set -eo pipefail

mlcp_opts="-database alex-test -host localhost -username admin -password admin"

mlcp import $mlcp_opts \
     -input_file_path input-files \
     -input_file_type aggregates \
     -aggregate_record_element chapter \
     -output_collections /chapter \
     -output_uri_prefix /chapter/ \
     -output_uri_suffix .xml
```

Dabei werden alle Dateien im Ordner *`input-files`* importiert. Der Dateityp der hochzuladenen Daten ist mit *`aggregates`* angegeben. Das sind XML Daten.

▻ Mehr Infos zu den Kommandozeilen-Optionen befinden sich auf der entsprechenden Dokuseite[125] von MarkLogic.

122) http://mlproj.org/
123) https://github.com/marklogic/marklogic-contentpump
124) https://developer.marklogic.com/products/mlcp
125) https://docs.marklogic.com/guide/mlcp/import

Mit der Option `-aggregate_record_element` wird definiert, dass die Eingabe bzgl. des Elements `<chapter>` aufgesplittet werden soll. D.h. eine Datei mit folgendem Inhalt:

```
<test>
  <title>Test Datei</title>
  <chapter>
    <title>Test Kapitel 1</title>
    <content>Kapitel Inhalt 1</content>
  </chapter>
  <chapter>
    <title>Test Kapitel 2</title>
    <content>Kapitel Inhalt 2</content>
  </chapter>
  <chapter>
    <title>Test Kapitel 2</title>
    <content>Kapitel Inhalt 2</content>
  </chapter>
</test>
```

wird in drei Records aufgesplittet:

Auf der Konsole kann man sich das Ergebnis der `mlcp` Sitzung anschauen. Es wurden - wie gewünscht - drei XML Fragmente separat in die Collection gespeichert.

figure 20: Ergebnis einer MarkLogic Content Pump Sitzung

▶ Um in MarkLogic keine Speicherpobleme zu erzeugen empfielt es sich große Dokumente, die man nur "speichern" will mit der Option `-document_type binary` zu importieren. In diesem Zusammenhang ist ebenfalls die Option `-streaming true` interessant.

Ein weiterer wichtiger Punkt, der mir bei der Arbeit mit `mlcp` aufgefallen ist:

Kommt es zu Inkonsistenzen in der Datenhaltung, mag das daran liegen, dass in verschiedenen ***mlcp*** Sitzungen von der gleichen Datei (gleicher Dateiname im Filesystem) importiert wurde.

- Es ist darauf zu achten, dass die Dateinamen eindeutig sind. Das kann zum Beispiel durch die Vergabe einer eindeutige ID im Dateinamen geschehen. Auf der Dokuseite zu den *mlcp* Optionen steht dazu folgendes:
- *"If your aggregate URI id's are not unique, you can overwrite one document in your input set with another. Importing documents with non-unique URI id's from multiple threads can also cause deadlocks."*
- *"The generated URIs are unique across a single import operation, but they are not globally unique. For example, if you repeatedly import data from some file /tmp/data.csv, the generated URIs will be the same each time (modulo differences in the number of documents inserted by the job)"*

3.3.5.3 Deployment-Tools

Will man eine größere MarkLogic Applikation maintainen, dann wird man um einen Build- / Deployment-Manager nicht herumkommen. Denn gewöhnlich gilt es einen Entwicklungsserver, einen Testserver und einen Prod-Server zu pflegen.

ml-gradle[126] und ml-proj[127] sind solche Tools. Das eine basiert auf der populären Java-Buildchain gradle[128], das andere ist eine NodeJS Applikation vom gleichen Maintainer, wie die zuvor beschriebene EXPath Konsole on page 135.

Wie ich bei einer größeren Evaluierungsaufgabe aber herausfinden konnte, eignen sich beide Tools eher für Projekte, die von der grünen Wiese aus gestartet werden, weil sie auf Namenskonventionen in der Dateistruktur setzen.

Beide beanspruchen zwar für sich eine (fast) vollständige Konfigurierbarkeit der Projektstruktur, jedoch ist diese mit einigen Fallstricken behaftet, so dass ich eine Migration eines Bestandsprojekts leider nicht empfehlen kann.

126) https://github.com/marklogic-community/ml-gradle
127) http://mlproj.org/
128) https://gradle.org/

ml-gradle

ml-gradle basiert auf dem populären Java Build-Tool ml-gradle[129] . Mittels vordefinierter oder eigener Tasks werden JSON Dateien automatisiert erstellt, die als Payload für REST Calls auf MarkLogic fungieren. Dabei wird die MarkLogic REST API[130] bedient. Bei der Deployment-Aktion selbst wird schliesslich das Payload eingesammelt und abgesetzt.

Grds. ist diese Idee nicht verkehrt, es gibt aber nun mehrere Stellen, an denen die Konfiguration modifiziert werden kann - was sich bei fehlender Disziplin der Entwickler natürlich eher nachteilig auswirken könnte:

- Setzen vordefinierter Properties[131] in der Datei `gradle.properties`
- Mittels vordefinierter Create-Tasks[132] , z.B. `gradle mlNewDatabase`

  werden Skelett-Dateien erzeugt, die man manuell bzgl. fehlender Konfigurationsinformationen vervollständigt.
- Es gibt einen Token-basierten Mechanismus[133], mit dem vordefinierte Platzhalter in Konfigurationsdateien (Payload-Files) dynamisch ersetzt werden.
- Eigene Groovy-Tasks können implementiert werden. Diese werden z.B. in der Datei `build.gradle` eingebunden. Beispiel für einen einfachen Groovy-Task[134]

mlproj

mlproj ist eine NodeJS Applikation von Florent Georges[135]. Wie auch bei ml-gradle würde ich den Umzug einer bestehenden Projektstruktur auf mlproj aber nicht empfehlen, sondern damit ein Projekt von der grünen Wiese aus starten.

Während es bei ml-gradle darum geht, den Überblick über mehrere, teilweise automatisch generierte, Konfigurationsdateien und -optionen zu behalten, geschieht die Konfiguration bei mlproj in nur einer Datei `prod.json` bzw. `dev.json` , welche die Einstellungen einer Basiskonfiguration `base.json` übernimmt bzw. überschreibt.

Natürlich ist der Erweiterungsmechanismus nicht ganz so mächtig, wie bei ml-gradle - schliesslich fehlt eine ausgewachsene Build-Chain als Unterbau. Ein NodeJS Programmierer wird damit aber gut zurechtkommen. Der Quellcode wirkt aufgeräumt und das Projekt ist gut dokumentiert.

Beide Tools haben einen "Watch"-Modus, bei dem die Dateistruktur des Entwicklers überwacht wird. Sobald sich eine Datei ändert, wird diese nach MarkLogic hochgeladen. Als Beispiel für die umfangreichen Konfigurationsmöglichkeiten ist folgend der Docstring des "Watch"-Modus in mlproj wiedergegeben:

```
mlproj watch [-a srv|-b db] [-s src|-/ dir|-1 file] [what]

-a, --as, --server <srv> server, get its modules database
-b, --db, --database <db> target database
-B, --sys, --system-database <db> the name of the target system db
-s, --src, --source-set <dir> source set to watch
-/, --dir, --directory <dir> directory to watch
-1, --doc, --document <file> file to watch
<what> source set or directory to watch
```

129) https://github.com/marklogic-community/ml-gradle
130) https://docs.marklogic.com/guide/rest-dev
131) https://github.com/marklogic-community/ml-gradle/wiki/Property-reference
132) https://github.com/marklogic-community/ml-gradle/wiki/Generating-new-resources
133) https://github.com/marklogic-community/ml-gradle/wiki/Configuring-resources
134) https://github.com/marklogic-community/ml-gradle/wiki/Debugging-module-loading
135) http://fgeorges.org/

3.4 XSL-FO mit XSLT1.x

XSL-FO ist eine XML basierte Auszeichnungssprache, die von einem XSL-FO Prozessor, wie dem kommerziellen AntennaHouse Formatter[136] oder der freien Open Source Lösung Apache FOP[137] verarbeitet wird.

Ausgabe ist meistens PDF, aber auch andere Formate, wie das veraltete RTF (Rich Text Format) oder auch spezielle Hardware spezifische Formate, wie PCL[138], werden unterstützt.

XSL Entwickler

XSL-FO ist eine Wissenschaft für sich, wie die sehr umfangreiche Spezifikation[139] belegt. Deshalb nimmt der XSL-Entwickler (Stylesheet-Entwickler) meistens eine Sonderrolle in einer XML Company ein.

Bei der Berufswahl gilt es zu beachten, dass ein "Stylesheet-Entwickler" in einer XML Company etwas ganz anderes macht, als ein "Stylesheet Spezialist" in einer Web Company. Ausserdem ist der XML Anwendungsbereich in zwei Sparten aufgeteilt:

- Publishing
- Datenkonvertierung

Beide Bereiche überlappen, und XML Entwickler werden sowohl im Publishing als auch in der Datenkonvertierung händeringend gesucht. Jedoch kommt ein Publisher seltener mit XML Datenbanken on page 103, und deren Abfragesprache XQuery als Programmiersprache on page 94 in Berührung. Dagegen wird sich der Konvertierer weniger mit kniffeligen Layoutproblemen herumschlagen müssen.

Diese Differenzierung ist wohl vergleichbar mit Backend- und Frontend-Entwicklung.

XSLT1.x

In der Version 1.x von XSLT musste man noch auf viele XPath Features verzichten. Bspw. wurde das Iterieren über eine Knotenmenge rekursiv mittels Template-Calls realisiert. Auch bzgl. der Arithmetik gab es noch nicht so viele Funktionen, wie man leicht anhand des Funktionskatalogs[140] überprüfen kann.

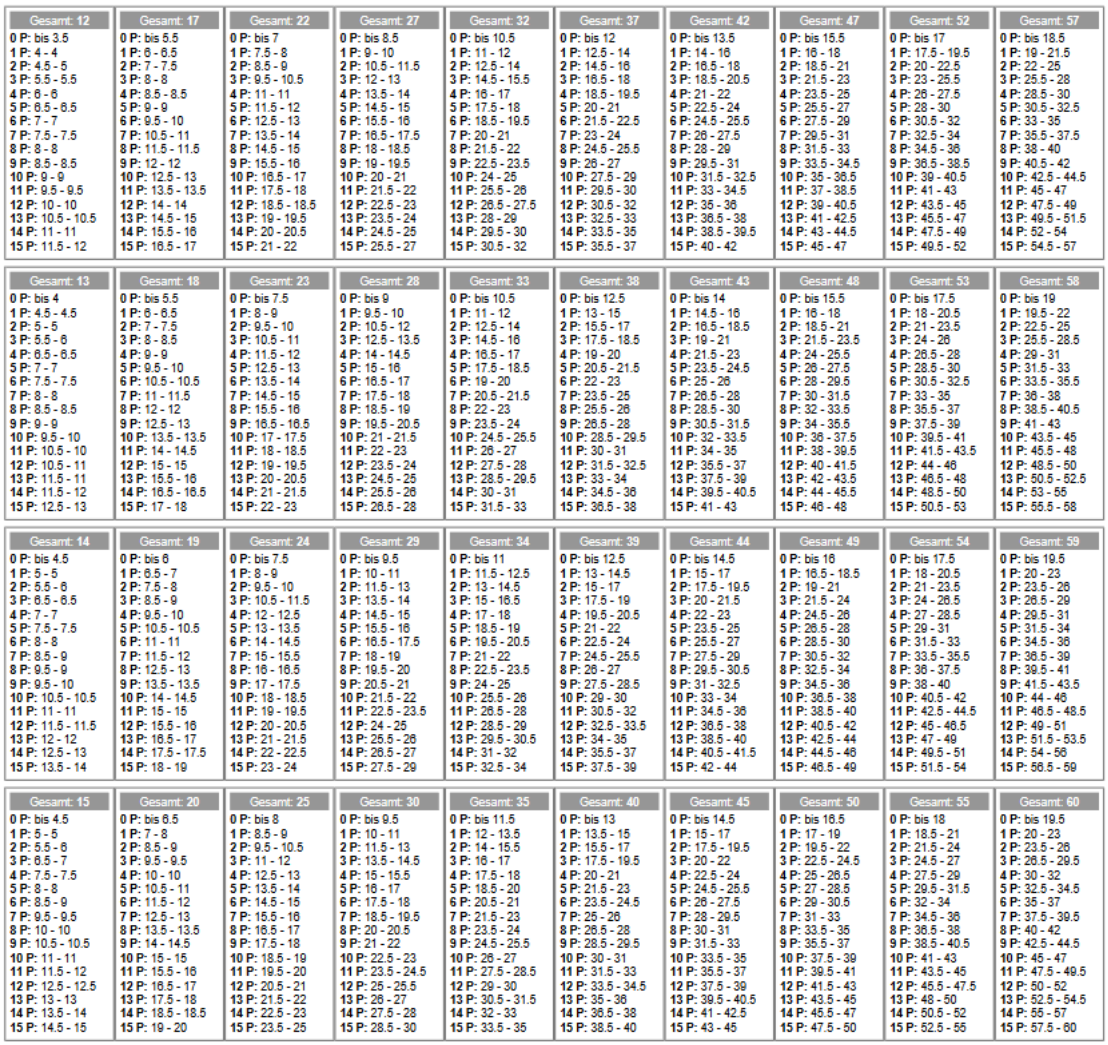
Vermutlich mag es also an den begrenzten Möglichkeiten liegen, warum mein damaliges Stylesheet (vor 10 Jahren) etwas komplizierter aussieht ...

136) https://www.antennahouse.com/formatter/
137) https://xmlgraphics.apache.org/fop/
138) https://en.wikipedia.org/wiki/Printer_Command_Language
139) https://www.w3.org/TR/xsl/
140) https://www.w3.org/TR/2010/REC-xpath-functions-20101214/

Notentabellen

Folgend ist eine Notentabelle zur Benotung nach dem 15-Punkte-Schema für die Kollegstufe abgebildet.

Notentabelle zur Benotung nach dem 15-Punkte Schema am Gymnasium. Mit ein paar Zeilen Code liessen sich schon in der Version 1.0 der Programmiersprache XSLT in Verbindung mit der Auszeichnungssprache XSL-FO beeindruckende PDF Layouts erzeugen.



Gesamt: 12	Gesamt: 17	Gesamt: 22	Gesamt: 27	Gesamt: 32	Gesamt: 37	Gesamt: 42	Gesamt: 47	Gesamt: 52	Gesamt: 57
0 P: bis 3.5	0 P: bis 5.5	0 P: bis 7	0 P: bis 8.5	0 P: bis 10.5	0 P: bis 12	0 P: bis 13.5	0 P: bis 15.5	0 P: bis 17	0 P: bis 18.5
1 P: 4 - 4	1 P: 6 - 6.5	1 P: 7.5 - 8	1 P: 9 - 10	1 P: 11 - 12	1 P: 12.5 - 14	1 P: 14 - 16	1 P: 16 - 18	1 P: 17.5 - 19.5	1 P: 19 - 21.5
2 P: 4.5 - 5	2 P: 7 - 7.5	2 P: 8.5 - 9	2 P: 10.5 - 11.5	2 P: 12.5 - 14	2 P: 14.5 - 16	2 P: 16.5 - 18	2 P: 18.5 - 21	2 P: 20 - 22.5	2 P: 22 - 25
3 P: 5.5 - 5.5	3 P: 8 - 8	3 P: 9.5 - 10.5	3 P: 12 - 13	3 P: 14.5 - 15.5	3 P: 16.5 - 18	3 P: 18.5 - 20.5	3 P: 21.5 - 23	3 P: 23 - 25.5	3 P: 25.5 - 28
4 P: 6 - 6	4 P: 8.5 - 8.5	4 P: 11 - 11	4 P: 13.5 - 14	4 P: 16 - 17	4 P: 18.5 - 19.5	4 P: 21 - 22	4 P: 23.5 - 25	4 P: 26 - 27.5	4 P: 28.5 - 30
5 P: 6.5 - 6.5	5 P: 9 - 9	5 P: 11.5 - 12	5 P: 14.5 - 15	5 P: 17.5 - 18	5 P: 20 - 21	5 P: 22.5 - 24	5 P: 25.5 - 27	5 P: 28 - 30	5 P: 30.5 - 32.5
6 P: 7 - 7	6 P: 9.5 - 10	6 P: 12.5 - 13	6 P: 15.5 - 16	6 P: 18.5 - 19.5	6 P: 21.5 - 22.5	6 P: 24.5 - 25.5	6 P: 27.5 - 29	6 P: 30.5 - 32	6 P: 33 - 35
7 P: 7.5 - 7.5	7 P: 10.5 - 11	7 P: 13.5 - 14	7 P: 16.5 - 17.5	7 P: 20 - 21	7 P: 23 - 24	7 P: 26 - 27.5	7 P: 29.5 - 31	7 P: 32.5 - 34	7 P: 35.5 - 37.5
8 P: 8 - 8	8 P: 11.5 - 11.5	8 P: 14.5 - 15	8 P: 18 - 18.5	8 P: 21.5 - 22	8 P: 24.5 - 25.5	8 P: 28 - 29	8 P: 31.5 - 33	8 P: 34.5 - 36	8 P: 38 - 40
9 P: 8.5 - 8.5	9 P: 12 - 12	9 P: 15.5 - 16	9 P: 19 - 19.5	9 P: 22.5 - 23.5	9 P: 26 - 27	9 P: 29.5 - 31	9 P: 33.5 - 34.5	9 P: 36.5 - 38.5	9 P: 40.5 - 42
10 P: 9 - 9	10 P: 12.5 - 13	10 P: 16.5 - 17	10 P: 20 - 21	10 P: 24 - 25	11 P: 25.5 - 26	11 P: 33 - 34.5	10 P: 35 - 36.5	10 P: 39 - 40.5	10 P: 42.5 - 44.5
11 P: 9.5 - 9.5	11 P: 13.5 - 13.5	11 P: 17.5 - 18	11 P: 21.5 - 22	11 P: 25.5 - 26	12 P: 26.5 - 27.5	12 P: 35 - 36	11 P: 37 - 38.5	11 P: 41 - 43	11 P: 45 - 47
12 P: 10 - 10	12 P: 14 - 14	12 P: 18.5 - 18.5	12 P: 22.5 - 23	12 P: 26.5 - 27.5	13 P: 28 - 29	13 P: 36.5 - 38	12 P: 39 - 40.5	12 P: 43.5 - 45	12 P: 47.5 - 49
13 P: 10.5 - 10.5	13 P: 14.5 - 15	13 P: 19 - 19.5	13 P: 23.5 - 24	13 P: 28 - 29	14 P: 29.5 - 30	14 P: 38.5 - 39.5	13 P: 41 - 42.5	13 P: 45.5 - 47	13 P: 49.5 - 51.5
14 P: 11 - 11	14 P: 15.5 - 16	14 P: 20 - 20.5	14 P: 24.5 - 25	14 P: 29.5 - 30	15 P: 30.5 - 32	15 P: 40 - 42	14 P: 43 - 44.5	14 P: 47.5 - 49	14 P: 52 - 54
15 P: 11.5 - 12	15 P: 16.5 - 17	15 P: 21 - 22	15 P: 25.5 - 27	15 P: 30.5 - 32	15 P: 35.5 - 37	15 P: 40 - 42	15 P: 45 - 47	15 P: 49.5 - 52	15 P: 54.5 - 57

Gesamt: 13	Gesamt: 18	Gesamt: 23	Gesamt: 28	Gesamt: 33	Gesamt: 38	Gesamt: 43	Gesamt: 48	Gesamt: 53	Gesamt: 58
0 P: bis 4	0 P: bis 5.5	0 P: bis 7.5	0 P: bis 9	0 P: bis 10.5	0 P: bis 12.5	0 P: bis 14	0 P: bis 15.5	0 P: bis 17.5	0 P: bis 19
1 P: 4.5 - 4.5	1 P: 6 - 6.5	1 P: 8 - 9	1 P: 9.5 - 10	1 P: 11 - 12	1 P: 13 - 15	1 P: 14.5 - 16	1 P: 16 - 18	1 P: 18 - 20.5	1 P: 19.5 - 22
2 P: 5 - 5	2 P: 7 - 7.5	2 P: 9.5 - 10	2 P: 10.5 - 12	2 P: 12.5 - 14	2 P: 15.5 - 17	2 P: 16.5 - 18.5	2 P: 18.5 - 21	2 P: 21 - 23.5	2 P: 22.5 - 25
3 P: 5.5 - 6	3 P: 8 - 8.5	3 P: 10.5 - 11	3 P: 12.5 - 13.5	3 P: 14.5 - 16	3 P: 17.5 - 18.5	3 P: 19 - 21	3 P: 21.5 - 23.5	3 P: 24 - 26	3 P: 25.5 - 28.5
4 P: 6.5 - 6.5	4 P: 9 - 9	4 P: 11.5 - 12	4 P: 14 - 14.5	4 P: 16.5 - 17	4 P: 19 - 20	4 P: 21.5 - 23	4 P: 24 - 25.5	4 P: 26.5 - 28	4 P: 29 - 31
5 P: 7 - 7	5 P: 9.5 - 10	5 P: 12.5 - 13	5 P: 15 - 16	5 P: 17.5 - 18.5	5 P: 20.5 - 21.5	5 P: 23.5 - 24.5	5 P: 26 - 27.5	5 P: 28.5 - 30	5 P: 31.5 - 33
6 P: 7.5 - 7.5	6 P: 10.5 - 10.5	6 P: 13.5 - 14	6 P: 16.5 - 17	6 P: 19 - 20	6 P: 22 - 23	6 P: 25 - 26	6 P: 28 - 29.5	6 P: 30.5 - 32.5	6 P: 33.5 - 35.5
7 P: 8 - 8	7 P: 11 - 11.5	7 P: 14.5 - 15	7 P: 17.5 - 18	7 P: 20.5 - 21.5	7 P: 23.5 - 25	7 P: 26.5 - 28	7 P: 30 - 31.5	7 P: 33 - 35	7 P: 36 - 38
8 P: 8.5 - 8.5	8 P: 12 - 12	8 P: 15.5 - 16	8 P: 18.5 - 19	8 P: 22 - 23	8 P: 25.5 - 26	8 P: 28.5 - 30	8 P: 32 - 33.5	8 P: 35.5 - 37	8 P: 38.5 - 40.5
9 P: 9 - 9	9 P: 12.5 - 13	9 P: 16.5 - 16.5	9 P: 19.5 - 20.5	9 P: 23.5 - 24	9 P: 26.5 - 28	9 P: 30.5 - 31.5	9 P: 34 - 35.5	9 P: 37.5 - 39	9 P: 41 - 43
10 P: 9.5 - 10	10 P: 13.5 - 13.5	10 P: 17 - 17.5	10 P: 21 - 21.5	10 P: 24.5 - 25.5	10 P: 28.5 - 29.5	10 P: 32 - 33.5	10 P: 36 - 37.5	10 P: 39.5 - 41	10 P: 43.5 - 45
11 P: 10.5 - 10	11 P: 14 - 14.5	11 P: 18 - 18.5	11 P: 22 - 23	11 P: 26 - 27	11 P: 30 - 31	11 P: 34 - 35	11 P: 38 - 39.5	11 P: 41.5 - 43.5	11 P: 45.5 - 48
12 P: 10.5 - 11	12 P: 15 - 15	12 P: 19 - 19.5	12 P: 23.5 - 24	12 P: 27.5 - 28	12 P: 31.5 - 32.5	12 P: 35.5 - 37	12 P: 40 - 41.5	12 P: 44 - 46	12 P: 48.5 - 50
13 P: 11.5 - 11	13 P: 15.5 - 16	13 P: 20 - 20.5	13 P: 24.5 - 25	13 P: 28.5 - 29.5	13 P: 33 - 34	13 P: 37.5 - 39	13 P: 42 - 43.5	13 P: 46.5 - 48	13 P: 50.5 - 52.5
14 P: 11.5 - 12	14 P: 16.5 - 16.5	14 P: 21 - 21.5	14 P: 25.5 - 26	14 P: 30 - 31	14 P: 34.5 - 36	14 P: 39.5 - 40.5	14 P: 44 - 45.5	14 P: 48.5 - 50	14 P: 53 - 55
15 P: 12.5 - 13	15 P: 17 - 18	15 P: 22 - 23	15 P: 26.5 - 28	15 P: 31.5 - 33	15 P: 36.5 - 38	15 P: 41 - 43	15 P: 46 - 48	15 P: 50.5 - 53	15 P: 55.5 - 58

Gesamt: 14	Gesamt: 19	Gesamt: 24	Gesamt: 29	Gesamt: 34	Gesamt: 39	Gesamt: 44	Gesamt: 49	Gesamt: 54	Gesamt: 59
0 P: bis 4.5	0 P: bis 6	0 P: bis 7.5	0 P: bis 9.5	0 P: bis 11	0 P: bis 12.5	0 P: bis 14.5	0 P: bis 16	0 P: bis 17.5	0 P: bis 19.5
1 P: 5 - 5	1 P: 6.5 - 7	1 P: 8 - 9	1 P: 10 - 11	1 P: 11.5 - 12.5	1 P: 13 - 14.5	1 P: 15 - 17	1 P: 16.5 - 18.5	1 P: 18 - 20.5	1 P: 20 - 23
2 P: 5.5 - 6	2 P: 7.5 - 8	2 P: 9.5 - 10	2 P: 11.5 - 13	2 P: 13 - 14.5	2 P: 15 - 17	2 P: 17.5 - 19.5	2 P: 19 - 21	2 P: 21 - 23.5	2 P: 23.5 - 26
3 P: 6.5 - 6.5	3 P: 8.5 - 9	3 P: 10.5 - 11.5	3 P: 13.5 - 14	3 P: 15 - 16.5	3 P: 17.5 - 19	3 P: 20 - 21.5	3 P: 21.5 - 24	3 P: 24 - 26.5	3 P: 26.5 - 29
4 P: 7 - 7	4 P: 9.5 - 10	4 P: 12 - 12.5	4 P: 14.5 - 15	4 P: 17 - 18	4 P: 19.5 - 20.5	4 P: 22 - 23	4 P: 24.5 - 26	4 P: 27 - 28.5	4 P: 29.5 - 31
5 P: 7.5 - 7.5	5 P: 10.5 - 10.5	5 P: 13 - 13.5	5 P: 15.5 - 16	5 P: 18.5 - 19	5 P: 21 - 22	5 P: 23.5 - 25	5 P: 26.5 - 28	5 P: 29 - 31	5 P: 31.5 - 34
6 P: 8 - 8	6 P: 11 - 11	6 P: 14 - 14.5	6 P: 16.5 - 17.5	6 P: 19.5 - 20.5	6 P: 22.5 - 24	6 P: 25.5 - 27	6 P: 28.5 - 30	6 P: 31.5 - 33	6 P: 34.5 - 36
7 P: 8.5 - 9	7 P: 11.5 - 12	7 P: 15 - 15.5	7 P: 18 - 19	7 P: 21 - 22	7 P: 24.5 - 25.5	7 P: 27.5 - 29	7 P: 30.5 - 32	7 P: 33.5 - 35.5	7 P: 36.5 - 39
8 P: 9.5 - 9	8 P: 12.5 - 13	8 P: 16 - 16.5	8 P: 19.5 - 20	8 P: 22.5 - 23.5	8 P: 26 - 27	8 P: 29.5 - 30.5	8 P: 32.5 - 34	8 P: 36 - 37.5	8 P: 39.5 - 41
9 P: 9.5 - 10	9 P: 13.5 - 13.5	9 P: 17 - 17.5	9 P: 20.5 - 21	9 P: 24 - 25	9 P: 27.5 - 28.5	9 P: 31 - 32.5	9 P: 34.5 - 36	9 P: 38 - 40	9 P: 41.5 - 43.5
10 P: 10.5 - 10.5	10 P: 14 - 14.5	10 P: 18 - 18.5	10 P: 21.5 - 22	10 P: 25.5 - 26	10 P: 29 - 30	10 P: 33 - 34	10 P: 36.5 - 38	10 P: 40.5 - 42	10 P: 44 - 46
11 P: 11 - 11	11 P: 15 - 15	11 P: 19 - 19.5	11 P: 22.5 - 23.5	11 P: 26.5 - 28	11 P: 30.5 - 32	11 P: 34.5 - 36	11 P: 38.5 - 40	11 P: 42.5 - 44.5	11 P: 46.5 - 48.5
12 P: 11.5 - 11.5	12 P: 15.5 - 16	12 P: 20 - 20.5	12 P: 24 - 25	12 P: 28.5 - 29	12 P: 32.5 - 33.5	12 P: 36.5 - 38	12 P: 40.5 - 42	12 P: 45 - 46.5	12 P: 49 - 51
13 P: 12 - 12	13 P: 16.5 - 17	13 P: 21 - 21.5	13 P: 25.5 - 26	13 P: 29.5 - 30.5	13 P: 34 - 35	13 P: 38.5 - 40	13 P: 42.5 - 44	13 P: 47 - 49	13 P: 51.5 - 53.5
14 P: 12.5 - 13	14 P: 17.5 - 17.5	14 P: 22 - 22.5	14 P: 26.5 - 27	14 P: 31 - 32	14 P: 35.5 - 37	14 P: 40.5 - 41.5	14 P: 44.5 - 46	14 P: 49.5 - 51	14 P: 54 - 56
15 P: 13.5 - 14	15 P: 18 - 19	15 P: 23 - 24	15 P: 27.5 - 29	15 P: 32.5 - 34	15 P: 37.5 - 39	15 P: 42 - 44	15 P: 46.5 - 49	15 P: 51.5 - 54	15 P: 56.5 - 59

Gesamt: 15	Gesamt: 20	Gesamt: 25	Gesamt: 30	Gesamt: 35	Gesamt: 40	Gesamt: 45	Gesamt: 50	Gesamt: 55	Gesamt: 60
0 P: bis 4.5	0 P: bis 6.5	0 P: bis 8	0 P: bis 9.5	0 P: bis 11.5	0 P: bis 13	0 P: bis 14.5	0 P: bis 16.5	0 P: bis 18	0 P: bis 19.5
1 P: 5 - 5	1 P: 7 - 8	1 P: 8.5 - 9	1 P: 10 - 11	1 P: 12 - 13.5	1 P: 13.5 - 15	1 P: 15 - 17	1 P: 17 - 19	1 P: 18.5 - 21	1 P: 20 - 23
2 P: 5.5 - 6	2 P: 8.5 - 9	2 P: 9.5 - 10.5	2 P: 11.5 - 13	2 P: 14 - 15.5	2 P: 15.5 - 17	2 P: 17.5 - 19.5	2 P: 19.5 - 22	2 P: 21.5 - 24	2 P: 23.5 - 26
3 P: 6.5 - 7	3 P: 9.5 - 9.5	3 P: 11 - 12	3 P: 13.5 - 14.5	3 P: 16 - 17	3 P: 17.5 - 19.5	3 P: 20 - 22	3 P: 22.5 - 24.5	3 P: 24.5 - 27	3 P: 26.5 - 29.5
4 P: 7.5 - 7.5	4 P: 10 - 10	4 P: 12.5 - 13	4 P: 15 - 15.5	4 P: 17.5 - 18	4 P: 20 - 21	4 P: 22.5 - 24	4 P: 25 - 26.5	4 P: 27.5 - 29	4 P: 30 - 32
5 P: 8 - 8	5 P: 10.5 - 11	5 P: 13.5 - 14	5 P: 16 - 17	5 P: 18.5 - 20	5 P: 21.5 - 23	5 P: 24.5 - 25.5	5 P: 27 - 28.5	5 P: 29.5 - 31.5	5 P: 32.5 - 34.5
6 P: 8.5 - 9	6 P: 11.5 - 12	6 P: 14.5 - 15	6 P: 17.5 - 18	6 P: 20.5 - 21	6 P: 23.5 - 24.5	6 P: 26 - 27.5	6 P: 29 - 30.5	6 P: 32 - 34	6 P: 35 - 37
7 P: 9.5 - 9.5	7 P: 12.5 - 13	7 P: 15.5 - 16	7 P: 18.5 - 19.5	7 P: 21.5 - 23	7 P: 25 - 26	7 P: 28 - 29.5	7 P: 31 - 33	7 P: 34.5 - 36	7 P: 37.5 - 39.5
8 P: 10 - 10	8 P: 13.5 - 13.5	8 P: 16.5 - 17	8 P: 20 - 20.5	8 P: 23.5 - 24	8 P: 26.5 - 28	8 P: 30 - 31	8 P: 33.5 - 35	8 P: 36.5 - 38	8 P: 40 - 42
9 P: 10.5 - 10.5	9 P: 14 - 14.5	9 P: 17.5 - 18	9 P: 21 - 22	9 P: 24.5 - 25.5	9 P: 28.5 - 29.5	9 P: 31.5 - 33	9 P: 35.5 - 37	9 P: 38.5 - 40.5	9 P: 42.5 - 44.5
10 P: 11 - 11	10 P: 15 - 15	10 P: 18.5 - 19	10 P: 22.5 - 23	10 P: 26 - 27	10 P: 30 - 31	10 P: 33.5 - 35	10 P: 37.5 - 39	10 P: 41 - 43	10 P: 45 - 47
11 P: 11.5 - 12	11 P: 15.5 - 16	11 P: 19.5 - 20	11 P: 23.5 - 24.5	11 P: 27.5 - 28.5	11 P: 31.5 - 33	11 P: 35.5 - 37	11 P: 39.5 - 41	11 P: 43.5 - 45	11 P: 47.5 - 49.5
12 P: 12.5 - 12.5	12 P: 16.5 - 17	12 P: 20.5 - 21	12 P: 25 - 25.5	12 P: 29 - 30	12 P: 33.5 - 34.5	12 P: 37.5 - 39	12 P: 41.5 - 43	12 P: 45.5 - 47.5	12 P: 50 - 52
13 P: 13 - 13	13 P: 17.5 - 18	13 P: 21.5 - 22	13 P: 26 - 27	13 P: 30.5 - 31.5	13 P: 35 - 36	13 P: 39.5 - 40.5	13 P: 43.5 - 45	13 P: 48 - 50	13 P: 52.5 - 54.5
14 P: 13.5 - 14	14 P: 18.5 - 18.5	14 P: 22.5 - 23	14 P: 27.5 - 28	14 P: 32 - 33	14 P: 36.5 - 38	14 P: 41 - 42.5	14 P: 45.5 - 47	14 P: 50.5 - 52	14 P: 55 - 57
15 P: 14.5 - 15	15 P: 19 - 20	15 P: 23.5 - 25	15 P: 28.5 - 30	15 P: 33.5 - 35	15 P: 38.5 - 40	15 P: 43 - 45	15 P: 47.5 - 50	15 P: 52.5 - 55	15 P: 57.5 - 60

figure 21: Ausschnitt aus einer PDF XSL-FO Demo

(Link zum PDF mit der vollständigen Notentabelle[141])

Die Note 6 ist in der Kollegstufe am Gymnasium mit "0 Punkte" bezeichnet. Die Note 5 ist aufgeteilt auf die Stufen "1 Punkt", "2 Punkte" und "3 Punkte". Um die Note 5 zu erzielen muss mindestens ein Drittel der Gesamtpunktzahl erreicht werden. Die Note 4 ist unterteilt in "4 Punkte", "5 Punkte" und "6 Punkte". Um die Note 4 zu erreichen müssen mindestens 50% der Gesamtpunktzahl erzielt werden. Die restlichen 50% der Gesamtpunkte werden dann möglichst gleichverteilt auf die Noten "7 Punkte" bis "15 Punkte".

Die Note "15 Punkte" entspricht dabei der "Note 1 mit Stern". Es können halbe Punkte vergeben werden. Gesucht ist eine homogene Verteilung nach obigen Regeln einer

141) http://www.tekturcms.de/stylesheets/margits_noten.pdf

vorgegebenen Gesamtpunktzahl auf die 15 Notenstufen. Das Stylesheet generiert über 200 Tabellen mit verschiedene Werten für die Gesamtpunktzahl aufgeteilt auf 5 DIN A4 Seiten.

Die Logik dazu sieht folgendermassen aus:

```
<xsl:template name="note-runden">
  <xsl:param name="note"/>
  <xsl:variable name="rest" select="$note * 10 mod 10"/>
  <xsl:choose>
    <xsl:when test="$rest &gt; 2.5">
      <xsl:value-of select="floor(number($note))"/>
    </xsl:when>
    <xsl:otherwise>
      <xsl:value-of select="floor(number($note))-0.5"/>
    </xsl:otherwise>
  </xsl:choose>
</xsl:template>

<xsl:template name="noten-tabelle">
  <xsl:param name="von-punkte"/>
  <xsl:param name="bis-punkte"/>
  <xsl:if test="$von-punkte &lt; $bis-punkte+1">
    <xsl:variable name="note6" select="$von-punkte div 3"/>
    <xsl:variable name="note4" select="$von-punkte div 2"/>
    <xsl:variable name="note5-schrittweite" select="($note4 +(-$note6)) div 3"/>
    <xsl:variable name="schrittweite" select="$note4 div 12"/>
    <xsl:variable name="note-6">
      <xsl:call-template name="note-runden">
        <xsl:with-param name="note" select="$note6"/>
      </xsl:call-template>
    </xsl:variable>
    <xsl:variable name="note6-korrigiert">
      <xsl:choose>
        <xsl:when test="3*($note-6+0.5) &lt; $von-punkte">
          <xsl:value-of select="$note6+0.5"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="$note6"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:variable>
    <fo:block border="1pt solid grey" space-after="3pt"
              keep-together.within-column="always" padding="2pt">
    <fo:block text-align="center" background-color="#999" color="white"
              padding-bottom="0pt" padding-top="2pt" space-after="2pt">
      <xsl:text>Gesamt: </xsl:text>
      <fo:inline  font-weight="bold">
        <xsl:value-of select="$von-punkte"/>
      </fo:inline>
    </fo:block>
    <fo:block>
      <fo:inline font-weight="bold">
        <xsl:text>0 P: </xsl:text>
      </fo:inline>
      <xsl:text>bis </xsl:text>
      <xsl:call-template name="note-runden">
        <xsl:with-param name="note" select="$note6-korrigiert"/>
      </xsl:call-template>
    </fo:block>
    <xsl:call-template name="note5">
      <xsl:with-param name="i" select="0"/>
      <xsl:with-param name="note6" select="$note6-korrigiert"/>
      <xsl:with-param name="note5-schrittweite" select="$note5-schrittweite"/>
      <xsl:with-param name="gesamt" select="$von-punkte"/>
    </xsl:call-template>
    <xsl:call-template name="ab-note4">
```

```
      <xsl:with-param name="i" select="0"/>
      <xsl:with-param name="note4" select="$note4"/>
      <xsl:with-param name="schrittweite" select="$schrittweite"/>
    </xsl:call-template>
    <fo:block/>
    <fo:block/>
    </fo:block>
    <xsl:call-template name="noten-tabelle">
      <xsl:with-param name="von-punkte" select="$von-punkte+1"/>
      <xsl:with-param name="bis-punkte" select="$bis-punkte"/>
    </xsl:call-template>
  </xsl:if>
</xsl:template>

<xsl:template name="note5">
  <xsl:param name="i"/>
  <xsl:param name="note6"/>
  <xsl:param name="note5-schrittweite"/>
  <xsl:param name="gesamt"/>
  <xsl:if test="$i &lt; 3">
    <xsl:variable name="note-von-gerundet">
      <xsl:call-template name="note-runden">
        <xsl:with-param name="note" select="number($note6) +
                                    number($i)*number($note5-schrittweite)"/>
      </xsl:call-template>
    </xsl:variable>
    <xsl:variable name="von" select="$note-von-gerundet+0.5"/>
    <xsl:variable name="bis">
      <xsl:call-template name="note-runden">
        <xsl:with-param name="note" select="number($note6)+
                                  (number($i)+1)*number($note5-schrittweite)"/>
      </xsl:call-template>
    </xsl:variable>
    <xsl:variable name="bis-korrigiert">
      <xsl:choose>
        <xsl:when test="$i=2 and (2*$bis)=number($gesamt)">
          <xsl:value-of select="$bis+(-0.5)"/>
        </xsl:when>
        <xsl:otherwise>
          <xsl:value-of select="$bis"/>
        </xsl:otherwise>
      </xsl:choose>
    </xsl:variable>
    <fo:block>
      <fo:inline font-weight="bold">
        <xsl:value-of select="$i+1"/>
        <xsl:text> P: </xsl:text>
      </fo:inline>
      <xsl:value-of select="$von"/>
      <xsl:text> - </xsl:text>
      <xsl:value-of select="$bis-korrigiert"/>
    </fo:block>
    <xsl:call-template name="note5">
      <xsl:with-param name="i" select="$i+1"/>
      <xsl:with-param name="note6" select="$note6"/>
      <xsl:with-param name="note5-schrittweite" select="$note5-schrittweite"/>
      <xsl:with-param name="gesamt" select="$gesamt"/>
    </xsl:call-template>
  </xsl:if>
</xsl:template>

<xsl:template name="ab-note4">
  <xsl:param name="i"/>
  <xsl:param name="note4"/>
  <xsl:param name="schrittweite"/>
  <xsl:if test="$i &lt; 12">
    <xsl:variable name="note-von-gerundet">
```

```
      <xsl:call-template name="note-runden">
        <xsl:with-param name="note" select="number($note4) +
                                    number($i)*number($schrittweite)"/>
      </xsl:call-template>
    </xsl:variable>
    <xsl:variable name="von" select="$note-von-gerundet+0.5"/>
    <xsl:variable name="bis">
      <xsl:call-template name="note-runden">
        <xsl:with-param name="note" select="number($note4)+
                                            (number($i)+1)*number($schrittweite)"/>
      </xsl:call-template>
    </xsl:variable>
    <fo:block>
      <fo:inline font-weight="bold">
        <xsl:value-of select="$i+4"/>
        <xsl:text> P: </xsl:text>
      </fo:inline>
      <xsl:value-of select="$von"/>
      <xsl:text> - </xsl:text>
      <xsl:value-of select="if ($i=11) then ($bis+0.5) else $bis"/>
    </fo:block>
    <xsl:call-template name="ab-note4">
      <xsl:with-param name="i" select="$i+1"/>
      <xsl:with-param name="note4" select="$note4"/>
      <xsl:with-param name="schrittweite" select="$schrittweite"/>
    </xsl:call-template>
  </xsl:if>
</xsl:template>
```

Die drei Templates rufen sich so lange selbst auf, bis eine Abbruchbedingung erreicht ist. In den Templates, in denen die zwei Sonderfälle Note 4 und Note 5 behandelt werden, bestimmt die Abbruchbedingung eine Laufvariable `$i`, die jeweils mit 3 (1P + 2P) und 12 (3P + 4P + 5P) ihren Schwellenwert hat. Das Template `notentabelle` dispatched auf die Sonderfälle und kümmert sich selbst um die Noten 1 bis 3.

Ohne jetzt groß meine Logik von damals zu überprüfen und in Frage zu stellen, finde ich doch, dass sich XSLT in den letzten Jahren sehr gemausert hat. Musste man sich früher noch mit diversen Saxon Bugs bzgl. der XPath Auswertung herumschlagen, ist diese nun doch äusserst stabil und es gelingen auch komplizierte Prädikate und verschachtelte Selektoren auf Anhieb.

Eine Iteration mittels Endrekursion[142] ist heute in den meisten Fällen nicht mehr notwendig. Auch Templates, die nur eine Hilfsfunktion realisieren, wie das obige Template `note-runden`, werden besser mittels `xsl:function` umgesetzt, was auch innerhalb eines XPath Ausdrucks ausgewertet werden kann.

XSL-FO Seitenvorlage

Um diese Logik nun in eine XSL-FO Transformation einzubauen, wickelt man einfach das typische XSL-FO Gerüst herum:

```
<xsl:stylesheet version="1.0" xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
  xmlns:fo="http://www.w3.org/1999/XSL/Format"
  xmlns:html="http://www.w3.org/1999/xhtml">

  <xsl:output method="xml" indent="no"/>

  <!-- Logik von oben hier einfuegen -->

  <xsl:template name="noten">
    <fo:root xmlns:fo="http://www.w3.org/1999/XSL/Format" font-size="6pt">
      <fo:layout-master-set>
        <fo:simple-page-master master-name="my-page">
          <fo:region-body
```

142) https://de.wikipedia.org/wiki/Endrekursion

```
            margin="5mm"
            column-gap="2mm"
            column-count="10"/>
          <fo:region-start extent="5mm"/>
          <fo:region-end extent="5mm"/>
        </fo:simple-page-master>
      </fo:layout-master-set>
      <fo:page-sequence master-reference="my-page">
        <fo:flow flow-name="xsl-region-body">
          <fo:block font-size="12pt" color="#999" font-weight="bold"
                    span="all" text-align="center" line-height="20pt"
                    letter-spacing="5mm">
                    <xsl:text>Notentabellen</xsl:text>
          </fo:block>
          <xsl:call-template name="noten-tabelle">
            <xsl:with-param name="von-punkte">12</xsl:with-param>
            <xsl:with-param name="bis-punkte">261</xsl:with-param>
          </xsl:call-template>
        </fo:flow>
       </fo:page-sequence>
     </fo:root>
   </xsl:template>
</xsl:stylesheet>
```

In der Seitenvorlage `fo:simple-page-master` wird das Layout der Seite festgelegt. Im Seitenfluss `fo:page-sequence` wird diese Vorlage referenziert.

Dieses Beispiel zeigt schon die Flexibilität des Ansatzes: Man erstellt für eine Publikation eine Reihe von Seitenvorlagen für die unterschiedlichen Layoutbereiche und referenziert bei Bedarf. Dadurch wird Redundanz vermindert. Ausserdem können die Vorlagen in einer anderen Ausgabestrecke ggf. parameterisiert wiederverwendet werden.

Relativ neu ist die Auszeichnung von Layout und Design im PDF Bereich mittels CSS. Der Fachbegriff hierzu lautet "CSS for Paged Media". AntennaHouse Formatter unterstützt diese Technologie und erste Ausgabestrecken werden damit umgesetzt. Wie aber die Entwickler von AntennaHouse bestätigen (Auf der XML Prag 2019) wird XSL-FO bzgl. der Umsetzung komplexer Layouts seine Alleinstellung behalten.

3.5 Testing

In diesem Kapitel werden einige ausgewählte Themen zum Testing von XSLT und XQuery Programmen vorgestellt.

3.5.1 Validierung mit Schematron

Um die Korrektheit einer XML Instanz zu prüfen, gib es verschiedene Schemata, wie **XSD**, **RNG** oder **DTD** , welche der Parser beim Aufbau des DOM Baums heranzieht, vgl. Kapitel Schemata on page 27. Eine Validierung mit Apache Xerces könnte beispielsweise als Java Code folgendermaßen angestossen werden:

```
URL schemaFile = new URL("http://host:port/filename.xsd");
Source xmlFile = new StreamSource(new File("web.xml"));
SchemaFactory schemaFactory = SchemaFactory
    .newInstance(XMLConstants.W3C_XML_SCHEMA_NS_URI);
try {
  Schema schema = schemaFactory.newSchema(schemaFile);
  Validator validator = schema.newValidator();
  validator.validate(xmlFile);
  System.out.println(xmlFile.getSystemId() + " is valid");
} catch (SAXException e) {
  System.out.println(xmlFile.getSystemId() + " is NOT valid reason:" + e);
```

```
} catch (IOException e) {
```

Schema Dateien können aber auch in XML Editoren eingebunden werden, um schon während der Eingabe der XML Instanz die Korrektheit zu überprüfen.

Das geht einerseits[143)] über die Angabe des Doctypes in der XML Instanz, anderseits bieten auch alle Editoren die Möglichkeit ein bestimmtes Schema explizit auszuwählen, um gegen dieses auf Anforderung zu validieren.

Gilt es komplexere Businessregeln zu überprüfen, die über Syntax-, Konsistenz- und einfache Korrektheitschecks hinausgehen, empfiehlt sich eine Validierung mit Schematron Regeln.

Schematron ist XSLT

Bei einer Schematron Validierung wird eine XML Instanz mit Hilfe eines automatisch generierten XSLT Stylesheets überprüft. Dieses kontextabhängige Stylesheet wird aus einer in der Schematron Syntax vom Autor verfassten Regelbasis, die wiederum in XML vorliegt, über ein zweites XSLT Stylesheet generiert - Dieses zweite XSLT Stylesheet ist sozusagen das eigentliche Schematron Programm.

Das folgende Diagramm veranschaulicht die Vorgehensweise anhand eines Filter-Szenarios, bei dem ein XML Dokument mit einigen ungültigen Passagen in eine gefilterte Darstellung überführt wird.

143) http://www.heise.de

Einfacher Batch-Prozess zur Validierung mit Schematron und anschliessendem Filtern der Ergebnisse

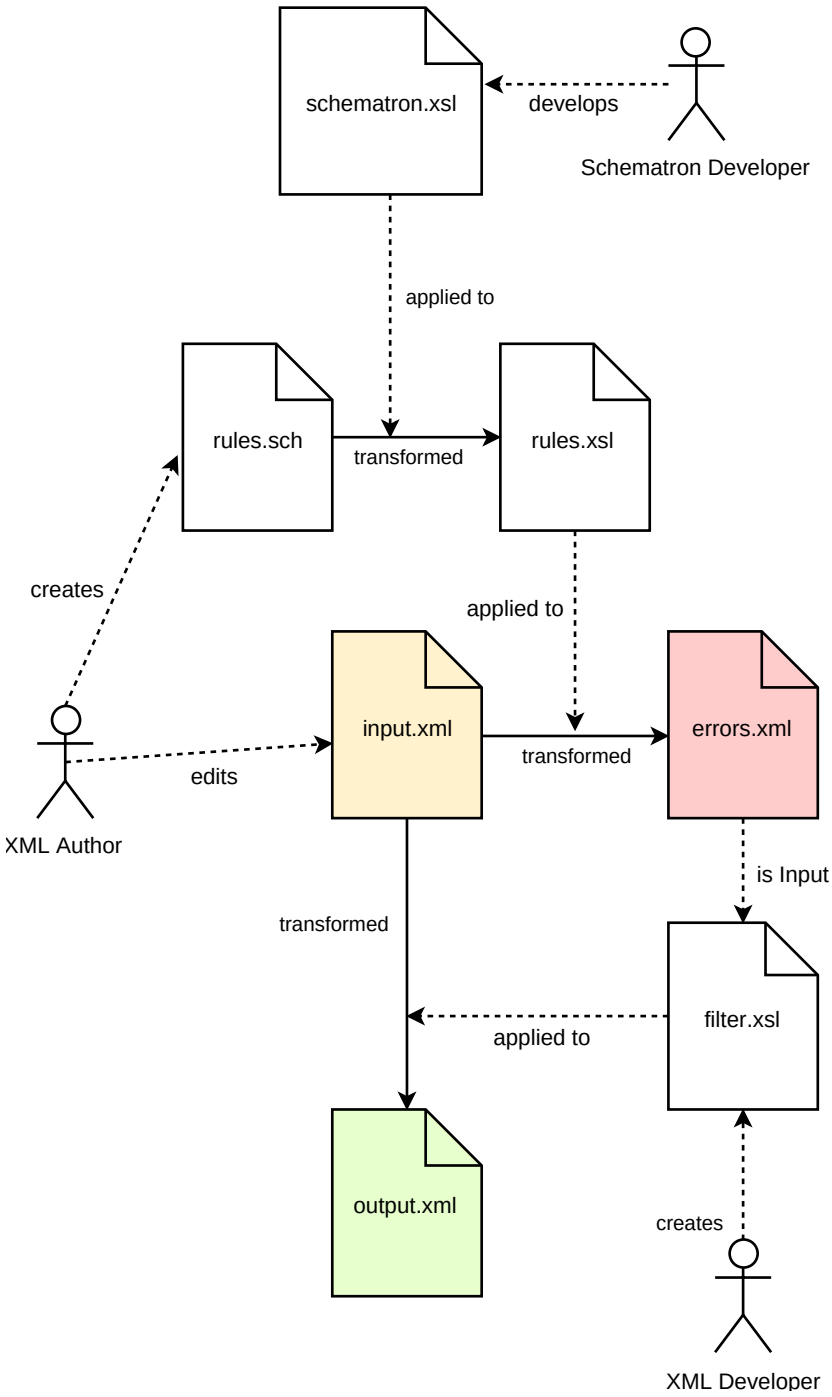


figure 22: Schematron Validierung mit Filter

Zu finden ist das Schematron Repo auf Github[144]. Dieses Repo ist etwas unübersichtlich. Der relevante Teil des Sourcecodes befindet sich unter: `schematron/code`

144) https://github.com/Schematron/schematron

CLI Verwendung

Um die Schematron XSLT Skripte in eine eigene XSLT Anwendung per Kommandozeile einzubinden, könnte man folgendermassen vorgehen:

- Im eigenen GIT Projekt das Schematron Projekt als Submodule referenzieren.
- Eine Regelbasis anlegen, beispielsweise `$project_name.sch` .
- Zwei Batch-Skripte anlegen, beispielsweise `generate_schema.sh` und `validate.sh` .

Mittels des Skripts `generate_schema.sh` wird aus der Schematron Regelbasis das Schematron XSLT Stylesheet generiert. Der Inhalt dieser Batchdatei könnte zum Beispiel so aussehen:

```
saxon $script_dir/$project_name_validation.sch $script_dir/schematron/iso_dsdl_include.xsl
 | \ saxon -s:- $script_dir/schematron/iso_abstract_expand.xsl | \
saxon -s:- $script_dir/schematron/iso_svrl_for_xslt2.xsl \
  generate-fired-rule=false > $script_dir/$project_name_validation.xsl
```

Der Prozess zum Erzeugen des projektspezifischen Validerungs-XSLT-Skripts ist dreistufig und wird über die folgenden XSLT Schritte abgearbeitet.

- `iso_dsdl_include.xsl`
- `iso_abstract_expand.xsl`
- `iso_svrl_for_xslt2.xsl`

Herauszufinden, was in diesen Skripten passiert, sei dem geneigten Leser selbst überlassen. Uns interessiert an dieser Stelle nur das Resultat, nämlich das XSLT Stylesheet `$project_name_validation.xsl` .

Dieses Skript wird in der Batchdatei `validate.sh` aufgerufen:

```
saxon $xml_instance_to_check.xml $script_dir/$project_name_validation.xsl \
  > $validation-result.xml
```

Die Ausgabe dieses Prüfprozesses ist eine XML Datei mit den Fehlern in der Eingabe-XML-Instanz, die weiterverarbeitet werden kann, beispielsweise als Filterkriterium für einen nachfolgenden Prozessschritt. Ihr Inhalt dieser Datei sieht z.B. wie folgt aus:

```
<svrl:schematron-output xmlns:svrl="http://purl.oclc.org/dsdl/svrl" [...]
  <svrl:active-pattern document="file:/Users/alex/xml_instance_to_check.xml"
   id="default" name="default"/>
  <svrl:failed-assert test="count(key('unique-ids', current()))=1">
    <svrl:text>ID is not unique!</svrl:text>
    <svrl:diagnostic-reference diagnostic="default">
      <bk:id xmlns:bk="http://tekturcms/namespaces/book">1234-5678-9</my:id>
    </svrl:diagnostic-reference>
  </svrl:failed-assert>
  [...]
```

Neben den `svrl:failed-assert` Elementen, die angeben, was bei der überprüften XML-Instanz fehlgeschlagen ist, gibt es auch die Möglichkeit sich positive Ergebnisse anzeigen zu lassen - über das Element `svrl:successful-report` .

Konkret sagt uns das obige XML Schnipsel, dass unsere `id` mit dem Wert `1234-5688-9` im geprüften XML Dokument nicht eindeutig ist. Die Schematron Regelbasis, die wir zur Überprüfung angegebenen haben, sieht so aus:

```
<schema xmlns:sch="http://purl.oclc.org/dsdl/schematron" [...]
  <xsl:key name="unique-ids" match="bk:id" use="."/>
  <sch:let name="date-regex" value="'^((19|2[0-9])[0-9]{2})-(0[1-9]|1[012])
                                    -(0[1-9]|[12][0-9]|3[01])$'"/>
  <sch:pattern id="default">
    <sch:rule context="book">
      <sch:assert id="check-book-id" role="error" test="count(key('unique-ids', bk:id))=1"
                  diagnostics="default">ID is not unique!</sch:assert>
      <sch:assert id="check-book-published" role="error"
                  test="matches(bk:published,$date-regex)
    </sch:rule>
    [...]
  </sch:pattern>
  <sch:diagnostics>
    <sch:diagnostic id="default">
      <xsl:element name="bk:id">
        <xsl:value-of select="bk:id"/>
      </xsl:element>
    </sch:diagnostic>
  </sch:diagnostics>
```

Neben der "successful" und "failed" Regeln ist auch die Deklaration von Funktionen und Variablen im Body der Regelbasis erlaubt. Dies ermöglicht komplexe Bedingungen, bespw. durch das Nachschlagen in einer Lookup-Tabelle abzuprüfen.

3.5.2 Erste Schritte mit XSpec

XSpec ist ein Test-Framework[145] für XSLT, XQuery und Schematron. Um beispielsweise komplexe Schematron Regeln zu testen, hinterlegt man in einem **Test-Szenario** Erwartungswerte für positive und negative Testfälle in Form von XML Schnippseln.

```
<test-szenario>
  <testfall>
    <personen>
      <person>
        <vorname>Horst</vorname>
        <nachname>Schlämmer</nachname>
        <gewicht>100</gewicht>
      </person>
      <person>
        <vorname>Gundula</vorname>
        <nachname></nachname>
        <gewicht>60</gewicht>
      </person>
    </personen>
  </testfall>
</test-szenario>
```

in einer XSpec Datei `*.xspec` werden **Assert- und Not-Assert-Methoden** deklariert:

```
<x:description xslt-version="2.0" xmlns:x="http://www.jenitennison.com/xslt/xspec"
               schematron="test.sch">
  <x:scenario label="ALL">
    <x:context href="test.xml"/>
    <x:expect-not-assert id="person-nachname-rule" location="//person[1]/nachname"/>
    <x:expect-assert id="person-nachname-rule" location="//person[2]/nachname"/>
  </x:scenario>
```

145) https://github.com/xspec

```
</x:description>
```

Grds. bdeutet ein Assert, dass das Mapping zwischen tatsächlichem Wert und Erwartungswert des Testfalls positiv erfüllt ist. Beim Not-Assert ist das Gegenteil der Fall. Im obigen Beispiel reichen zwei Regeln, um den Testfall vollständig abzudecken.

Wenn man Schematron Regeln mit Hilfe von XSpec testen will, dann muss man ein bisschen um die Ecke denken. Denn auch diese Regeln werden mittels Assert und Not-Assert modelliert.

```
<sch:schema xmlns:sch="http://purl.oclc.org/dsdl/schematron"
            xmlns:xsl="http://www.w3.org/1999/XSL/Transform" queryBinding="xslt2">
  <sch:pattern id="main">
    <sch:rule context="nachname">
      <sch:assert id="person-nachname-rule" role="error" test="normalize-space(.)">
        Der Nachname der Person mit ID: <sch:value-of select="@id"/> fehlt!
      </sch:assert>
    </sch:rule>
  </sch:pattern>
</sch:schema>
```

In der Schematron-Regel wird zugesichert (Assert), dass jede Person einen Nachnamen hat.

Hat sie keinen Nachnamen, so wird der Bericht zum Fehlerfall in die Schematron Ergebnisdatei geschrieben. Diese Datei wertet nun XSpec aus.

> Erscheint ein Fehler (= das Feld **nachname** ist leer), so greift bei XSpec die Assert-Regel! Das ist die umgekehrte Logik zu den Schematron Regeln.

Als Eselsbrücke kann man ein Assert in der XSpec Datei gleichsetzen mit **Appear** und ein Not-Assert mit **Not-Appear**.

Ein Assert sichert also zu, dass sich ein Fehlerbericht in der Schematron Ergebnisdatei zum Testfall befindet. Ein Not-Assert sichert zu, dass sich kein Fehlerbericht befindet.

Wie man sich leicht vorstellen kann, sind Assert-Regeln in diesem Fall leicht zu finden, dazu muss man nur die Schematron Testregeln ins Leere zeigen lassen. Alles ist grün und alles ist gut - dem Augenschein nach.

3.6 Performanz-Optimierung

Bei der Verarbeitung größerer Datenmengen, gibt es mehrere Optionen. Entweder man verlässt sich auf die Performanzoptimierung, die im XSLT- / XQuery- Prozessor implementiert ist, bzw. benutzt Methoden, wie XSLT Streaming on page 60 - oder man setzt eine Ebene darunter an und schaut zu, was Java so treibt.

Es ist natürlich auch möglich seine Algorithmen umzustricken, den Code zu optimieren und das ganze (noch) kryptischer zu machen. Aber ganz nach dem Motto *"Never Change a running system"* wird vielerorts zunächst einmal an der Basis gefeilt.

Heap Memory und Garbage Collector

Wenn ich mich recht an mein Informatikstudium erinnere, dann sammelt ein Garbage Collector automatisch Speicherblöcke ein, die das Java Programm (Saxon) nicht mehr braucht und übergibt diese an die Heap Memory Verwaltung. Ein Heap ist meistens ein automatisch balanzierender Baum, der die freigewordenen Speicherblöcke der Größe nach sortiert verwaltet.

Da auch der Garbage Collector zur selben Zeit läuft, wie unser Java Programm, können sich die beiden ins Gehege kommen. Hat man für das Java Programm zuviel Heap-Space reserviert, dann lagert der Garbage Collector seine temporären Erzeugnisse auf einen langsamen Swap-Bereich aus. Hat man dagegen zuwenig reserviert, dann ist nicht genügend Platz für das Hauptprogramm vorhanden.

Gebräuchliche Fehlermeldungen sind z.B.:

- `java.lang.OutOfMemoryError: Java heap space`
- `java.lang.OutOfMemoryError: GC Overhead limit exceeded`

Mehr dazu steht auf der Oracle Webseite zum Thema[146)]

Diese viel diskutierten Heap Einstellungen der Java Virtual Machine (JVM), wirken sich besonders bei Applikationen aus, bei denen der Speicherverbrauch zu einer bestimmten Zeit nicht vorhersehbar ist, z.b. bei einem Java-Game mit Action- und auch mal Rollenspielelementen.

Hier wäre ein Tuning bzgl. der JVM Heap Optionen angebracht, zumal man das Spiel auch für Spieler optimieren will, die nicht auf die leistungsfähigste Hardware zugreifen können.

Gängige Optionen, die man beim Java-Aufruf mitgibt, sind bspw.

- `Xms<size> set initial Java heap size`
- `Xmx<size> set maximum Java heap size`
- `Xss<size> set java thread stack size`

Mehr dazu steht auf der Oracle Webseite zum Thema[147)]

Für die batch-orientierten Prozesse bei der XML Verarbeitung sind diese Optionen meist nicht ausschlaggebend, da der Heap bei der Transformation eines grösseren Dokuments linear belastet wird. Hier ist es sogar so, dass man das obere Limit für den Heap-Speicher mit der unteren Schranke gleichsetzen sollte, um der JVM die Auswahl des geeigneten Speicherbereichs zur Laufzeit zu ersparen.

Ich benutze für meine performanzlastigen Transformationen eine 10 GB Heap Schranke:

```
-Xms10g -Xmx10g
```

Damit lassen sich auch sehr große XML Dateien mit einer Pipeline, wie MorganaX-Proc[148)] , die die Ergebnisdokumente der einzelnen Transformationsschritte im Speicher behalten, effizient verarbeiten.

146) https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/memleaks002.html
147) https://www.oracle.com/technetwork/java/gc-tuning-5-138395.html
148) https://www.xml-project.com/morganaxproc/

4 Zusätzliches Know-How

Contents

Unsortierte Notizen, die für jeden XSLT-Programmierer interessant sein könnten.

4.1 XML Editoren

Der XSLT Styesheet-Entwickler wird sich gewöhnlich mit Eingabedaten beschäftigen, die entweder automatisch mittels irgendeines Prozesses erzeugt wurden, oder die durch einen menschlichen Autor mit einem XML Editor eingegeben wurden.

Aus diesem Grund ist es ganz nützlich, die wichtigsten Editoren zu kennen.

Wir unterscheiden zwischen Desktopapplikationen und Webanwendungen. Ausserdem unterscheiden wir noch, ob der Editor WYSIWIG (**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **G**et) oder WYSIWYM (**W**hat **Y**ou **S**ee **I**s **W**hat **Y**ou **M**ean) unterstützt, oder ob er eine Mischung aus beidem darstellt. Das Buzzword hierzu heisst auch "Structured Content Authoring"

bla

WYSIWYM Desktop

Editor	Beschreibung
XMetal [XM]	XMetal ist wahrscheinlich der am weitesten verbreitete reine WYSIWYM Editor. Er hat Schnittstellen zu COM und Java und kann daher in eigene CMS integriert werden.
Arbortext XML Editor [EP]	Arbortext XML Editor, früher bekannt als EPIC ist sehr betagt. Bekanntermassen ist sein Tabelleneditor etwas buggy.

[XM] https://xmetal.com/
[EP] https://www.ptc.com/en/products/service-lifecycle-management/arbortext/editor

WYSIWYG Desktop

XMetal kann so konfiguriert werden, dass bei einer einfachen DTD der Content Bereich wie Word aussieht. Auch Code Editoren, wie OxygenXML bieten diese Möglichkeit. Das Key-Handling bei dieser Variante zeigt aber schnell, dass die UX noch weit von herkömmlichen Textverarbeitungssystem, wie Word oder OpenOffice entfernt ist.

WYSIWYM Online

Editor	Beschreibung
Oxygen XML WebAuthor [OX]	Dieser Online-Editor verwendet auf der Serverseite dieselbe Logik, wie das Desktop Programm des Herstellers. Das führt dazu, dass bei jedem Tastendruck eine Verbindung zum Server aufgebaut wird, und die Verarbeitung langsam werden kann. Zum Betrieb und bzgl. Customizing ist einschlägiges Java-Know-How erforderlich.
FontoXML [FX]	FontoXML sieht schon fast aus wie Word. Neben der WYSWYG/M Darstellung, kann auch die XML Struktur in einem Seitenpanel angezeigt werden.
XEditor [XE]	Xeditor benutzt XSLT Transformationen, um aus der Eingabe die Editoransicht zu generieren. Beim Abspeichern wird der umgekehrte Weg bestritten. Das mag zwar auf den ersten Blick etwas holprig erscheinen, wie aber auch Tektur beweist, funktioniert das ganze recht gut und schnell.
Xopus [XO]	Xopus ist wohl der älteste web-basierte XML Editor. Ich hatte damit schon 2008 zu tun, als er für ein Redaktionssystem evaluiert wurde. Wir haben uns dann für eine eigene nicht-generische Lösung basierend auf dem Webeditor CKEditor entschieden.

Das Customizing dieser Editoren erfordert einen sehr hohen Aufwand. Es müssen diverse Ressourcen angepasst werden, wie XSLT Skripte, XSD Schemas, CSS und Javascript. Das Schema wird meist über Kommandozeilentools in eine JS Repräsentation überführt.

Aus diesem Grund bieten einige Hersteller spezielle Schulungen an, wo man die Bedienung erlernen kann. Aus meiner Sicht ist das Problem "Webbasierter XML Editor" weltweit noch nicht ausreichend gelöst.

Die Kosten für den Betrieb rangieren um die 1000 EUR monatl. für ein 20 Benutzer-Setup.

4.2 Quellcode-Versionskontrolle

Um Zurückverfolgen zu können, was man an den Kunden ausgeliefert hat, welche Meilensteine auf dem Weg zum fertigen Produkt zurücklegt wurden - und ganz allgemein, um den Überblick über Quellcodeänderungen zu behalten, empfiehlt sich der Einsatz eines Versionskontrollsystems.

Mittlerweile ist **GIT** vom Linux Erfinder Linus Toralds das Mass aller Dinge. Je nach Anwendungsfall gibt es aber auch (historische) Alternativen.

[OX] https://www.oxygenxml.com/oxygen-xml-web-author/app/oxygen.html
[FX] https://www.fontoxml.com/
[XE] http://www.xeditor.com/portal
[XO] http://xopusfiddle.net/VT7T/3/

4.2.1 Kurze Geschichte zur Versionskontrolle Test

Ich vermute, dass das Problem der Versionsverwaltung von Source Code Dateien seit Anbeginn der Programmierung existiert. So ist es nicht verwunderlich, dass die Lösungen hierzu immer weiter verfeinert werden. Zu jeder Epoche gibt es allerdings nur einen Platzhirschen, der allen Entwicklern gleichermassen heilig ist.

RCS

Bis zu den 90er Jahren war das in Unix Systeme integrierte, RCS[155] beliebt. Jede Linux Distribution hat diese Lösung immer noch an Bord, und die Bedienung ist relativ einfach.

Bei RCS werden Änderungen an einzelnen Dateien in der jeweils zugeordneten Archivdatei verwaltet. Man kann dann die aktuelle Version in das Archiv einchecken und mit einer Logmeldung versehen.

Eine ausgecheckte Datei wird für andere Nutzer mit einem Lock versehen und gesperrt. Eine Check-In Sitzung sieht z.B. so aus:

```
$ ls -l
-rw-r--r--    1 Alex users          19 10. Oct 16:30 test.txt
$ cat test.txt
Hallo Welt!
$ ci -l test.txt
test.txt,v  <--  test.txt
enter description, terminated with single '.' or end of file:
NOTE: This is NOT the log message!
>> test check-in
>> .
initial revision: 1.1
done
$ ls -l
-rw-r--r--    1 alex users          19 10. Jan 14:25 test.txt
-r--r--r--    1 Alex users         218 10. Jan 14:25 test.txt,v
$ echo „bla bla blubb“ >> test.txt
$ ci -l test.txt
test.txt,v  <--  test.txt
new revision: 1.2; previous revision: 1.1
enter log message, terminated with single '.' or end of file:
```

Es gibt zusätzlich die Möglichkeit Branches anzulegen (über das Hinzufügen einer weiteren Stelle zur Versionsnummer).

Verschiedene Kopien können auch mittels einer Merge-Operation zusammengeführt werden. Ein Mehrbenutzerbetrieb ist also schon ansatzweise realisiert.

Praktisch ist RCS für mich nach-wie-vor, wenn ich eine elnzelne Datei trocken will, wie z.B. meinen Lebenslauf im ASCII-Text Format. Mittels Schlüsselwörter, wie `$Author$`, `$Date$` und `$Log$`, lassen sich Metadaten zur Version automatisch in die Arbeitsdatei übernehmen.

RCS, das also den Mehrbenutzebetrieb nur über Absprachen bzgl. des Locking-Mechanimsu erlaubt, wurde in den 90er Jahren von CVS abgelöst.

CVS

Wie der Name schon sagt (Concurrent Versions System), erlaubt CVS[156] den reibungslosen Mehrbenutzerbetrieb.

Im Gegensatz zu RCS werden Archivdateien nicht an ggf. verschiedenen frei definierbaren Orten gespeichert, sondern zentral in einem **Repository**.

155) https://de.wikipedia.org/wiki/Revision_Control_System
156) https://de.wikipedia.org/wiki/Concurrent_Versions_System

Mittels grafischer Clients und der Integration in alle gängigen IDEs (Integrated Development Environment), erlaubt CVS bspw. ein komfortables grafisches Diffing, was den Review und die Übernahme von Änderungen eines anderen Programmierers, wesentlich erleichtert.

Subversion

CVS hatte einige Unzulänglichkeiten, was das Handling von Binärdaten und Verzeichnissen betraf, sowie einige Sicherheitsmängel[157)]

Der Nachfolger von CVS ist Subversion[158)].

Was den Feature-Umgang angeht, sollte Subversion eigentlich für die meisten Anwendungsfälle in der Software-Entwicklung ausreichen, wäre da nicht die Open-Source Bewegung, die weltweit verteilt an Tausenden Projekten arbeitet.

So wurde das zentrale Repository, das eigentlich das Killer-Feature von CVS im Vergleich zu RCS war, nach der Jahrtausendwende mehr oder weniger über den Haufen geworfen.

Linus Toralds, der Erfinder von Linux, stellte GIT[159)] vor.

GIT

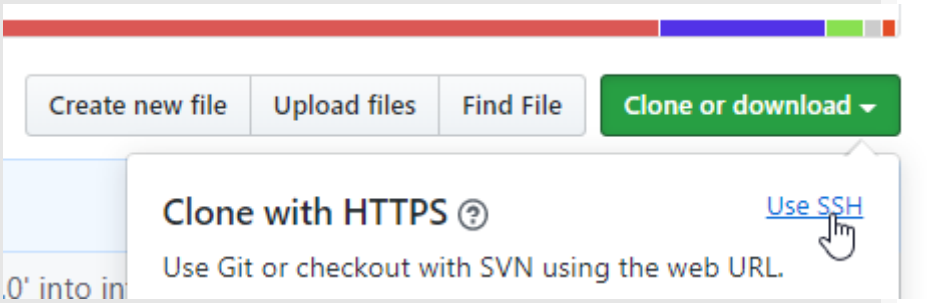
GIT[160)] weicht das Konzept des zentralen Repositories auf. Es gibt zwar meistens ein Repository, das zentral über das Netz erreichbar ist, jeder Entwickler arbeitet aber zusätzlich noch auf einer lokalen Kopie, die er regelmässig mit dem Haupt-Repository synchronisiert.

Ausserdem kann jedes Repository auch ge-„forkt“ werden, d.h. es wird von einer anderen Entwickler(-gruppe) kopiert, mit dem Ziel eine neue Variante der Software oder gar ein anderes Produkt zu erschaffen, was nach dem Open-Source Gedanken natürlich völlig legitim ist.

4.2.2 GIT Kommandos

Die wichtigsten GIT Befehle - Rubrik "Note-to-self":

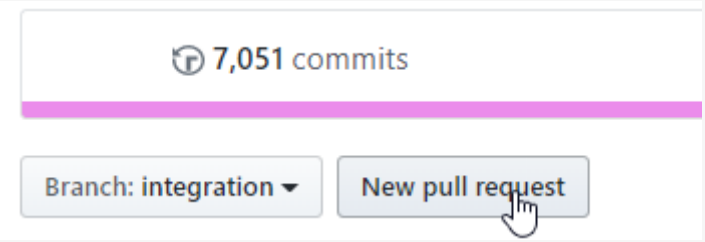
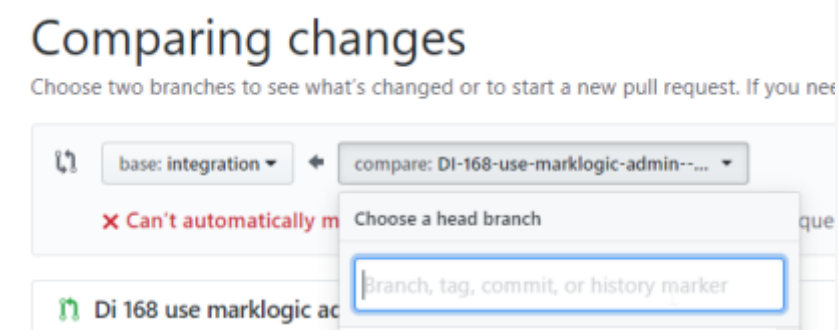
157) https://www.cvedetails.com/vulnerability-list/vendor_id-442/CVS.html
158) https://de.wikipedia.org/wiki/Apache_Subversion
159) https://de.wikipedia.org/wiki/Git
160) https://de.wikipedia.org/wiki/Git

Beschreibung	Kommando / Beispiel
Bestehendes GIT Repository clonen	`git clone https://git.tekturcms.de/tekturcms/tektur.git` ▶ Wenn man sich auf der Kommandozeile bewegt, dann kann es sein, dass man keinen Zugriff auf den HTTP Endpoint von GutHub hat, auf der GitHub Website kann man sich auch die URL für die SSH Verbindung geben lassen. 
GIT Repository clonen und alle GIT Submodule gleichzeitig clonen	`git clone --recurse-submodules git://github.com/foo/bar.git`

Veränderung einer Datei zum GIT Tracking hinzufügen, damit diese beim nächsten Commit erfasst wird	`git add pfad/im/git/projekt/zur/datei.txt`
	Wildcards funktionieren auch
	`git add *`

Alle Veränderungen in das lokale, geklonte Repository einchecken	`git commit -m "Form validation added”`
	ohne die Dateien vorher explizit per [[code:add] zu registrieren
	`git commit -a -m "Form Validation added"`

Alle vorhandenen Branches im Remote Repository holen	`git fetch`
Einen bestimmten Branch auschecken	`git checkout "TEKTUR-experimental-branch"`
Einen Branch anlegen	`git branch "TEKTUR-experimental-branch"`
Alle vorhanden Branches auflisten	`git branch`
Lokale Änderungen auf den Remote Branch pushen	`git push origin "TEKTUR-experimental-branch"`

Pull Request auf GitHub anlegen	
Den Pull Request auf einen bestimmten Base-Branch beziehen	
Zum Vorgänger eines beliebigen Commits wechseln	`git checkout '67b7474a773c4d6f76dc0915b290400b313c0bf5^'` Hier ist das Dach-Zeichen wichtig, das angibt, dass der vorherige Commit ausgecheckt werden soll
Einen Commit rückgängig machen	`git revert '67b74'` *`git revert`* erzeugt einen neuen Commit, der den angegeben Commit rückgängig macht
Ausgecheckten Branch mit einem anderen Branch mergen	`git merge experimental`
Grafisches Tool starten	`gitk`
Lokale Änderungen rückgängig machen	`git stash`
Auf Version des Git Repos auf dem Server zurücksetzen	`git reset --hard origin/master`

Branch mit master aktualisieren	Im Branch:So wird ein neuer Commit erzeugt...
	`git merge origin/master`
	So erscheint kein neuer Commit...
	`git fetch`
	`git rebase origin/master`

Glossary

Test

horst

A	**Automatischer Satz**	Bei einer Printpublikation kommt es auf gute Wort-, Zeilen-, Spalten und Seitenumbrüche an. Der mittels XML ausgezeichnete Content wird von einem Formatierprozess diesbzgl. automatisch an die richtige Stelle *"gesetzt"*.
C	**Core-Stylesheet**	In einem Stylesheet-Projekt bezeichnet das Core-Stylesheet eine bereits ausgiebig getestete Variante, die mittels Sub-Stylesheet unter Ausnutzung der XSLT Import Präzedenz überschrieben wird.
	Content Delivery Portal	Die beim Single Source Publishing erzeugten Ausgabeformate werden über Webportale zielgerecht verteilt. Aktuell experimentiert man mit QR Codes an Maschinen, die über das Smartphone gescannt werden können und daraufhin die Doku zur Maschine herunterladen.
	CCMS	*Component Content Management System* - Um Konzepte wie *Topic Based Authoring* realisieren zu können, sind spezielle Content Management Systeme erforderlich, die nicht *"Kapitelbasiert"* verwalten, sondern feingranular Topics und die Beziehungen zwischen diesen.
D	**DITA**	DITA ist ein Informationsmodell für die Technische Dokumentation - es unterstützt *Topic Based Authoring*.
	Diffing	Beim *Diffing* werden zwei XML Instanzen miteinander verglichen. Ein Diffing Prozessor markiert hingefügte, gelöschte und veränderte XML Elemente in einer *Diffing-Instanz*.
G	**Gültigkeiten**	In einer technischen Publikation werden Satzbausteine und Grafiken mit Gültigkeiten versehen, um bspw. länderspezifische Produktvarianten zu kennzeichnen.
I	**Intelligente Querverweise**	Links zwischen Textstellen in verschiedenen Topics und Publikationen bleiben versionstreu, d.h. in einer früheren Version der Publikation werden diejenigen Kapitel und Texte referenziert, die zum Zeitpunkt der Publikation aktuell waren.
P	**Parameterisierung**	Bei der Parameterisierung wird ein bestehendes Stylesheet mit Parametern versehen, um für möglichst viele Produktvarianten und Ausgabeformate die gleiche Codebasis wiederverwenden zu können. Dadurch soll Redundanz eingespart werden und der Aufruf vereinfacht werden.
	Push- vs. Pull	Push und Pull sind zwei Konzepte, wie man Transformationen mit XSLT gestalten kann. Die einen "ziehen"

		Informationen aus der Eingabe und setzen sie an die richtige Stelle in der Ausgabe. Die anderen transformieren die Eingabe schrittweise in die Ausgabe, vgl. Kapitel Push vs. Pull Stylesheets (imported) 19. Mischformen sind die Regel.
S	**Sub-Stylesheet**	Ein Sub-Stylesheet spezialisiert das Core-Stylesheet, damit Redundanz vermieden wird und somit die Wartbarkeit gewährleistet werden kann.
	SGML	SGML ist der Vorläufer von XML.
	Single Source Publishing	Beim Single Source Publishing wird aus einer XML Quelle eine Vielzahl von Ausgabeformaten erzeugt
	Structured Content Authoring	Der Content wird beim *Structured Content Authoring* semantisch mittels XML Tags ausgezeichnet. Bei einem WYSIWYG Ansatz sind die meisten Tags nur optional sichtbar.
T	**Topic Based Authoring**	Beim Topic Based Authoring steht nicht das gesamte Buch im Vordergrund, sondern der Inhalt wird in wiederverwendbare Topics aufgeteilt, die dann in verschiedenen Publikationen referenziert werden können.
	TIOBE Index	Im TIOBE Index wird jährlich die Beliebtheit von Programmiersprachen erfasst.
U	**Übersetzungsmanagement**	Damit ist die Verwaltung der Übersetzungen einer tech. Publikation gemeint. Da eine größere Publikation ständig aktualisiert wird, die Übersetzungen aber von externen Übersetzungsbüros einige Zeit in Anspruch nehmen, ist eine gewisse Abstimmungslogik erforderlich.
V	**Vortransformation**	Ein Schritt in der Prozesskette einer XSLT Transformation, der die Daten für den nächsten Schritt vorbereitet, vgl. Kapitel Vortransformationen (imported) 50.
X	**XML Datenbanken**	XML Datenbanken sind NoSQL Datenbanken, d.h. "Not only SQL" oder auch tatsächlich "No SQL" wird unterstützt. Die Spezialisierung erfolgt auf XML Daten.

Index (Fig.)

Footnotes

1 https://de.wikipedia.org/wiki/Darwin_Information_Typing_Architecture
DITA ist ein Standard im Bereich Publishing und löst ältere Dokumenttypen, wie z.B. Docbook ab. Beispielsweise ist DITA ein gutes Modell für Softwarehandbücher und zugehörige Online-Dokumentation., page 3

2 https://de.wikipedia.org/wiki/WYSIWYG
What You See Is What You Get - Mit dieser Eingabemethode hat der Autor schon ein Bild davon, wie sein Text gedruckt werden kann., page 3

3 https://en.wikipedia.org/wiki/Topic-based_authoring
Beim Topic Based Authoring wird der Content feingranular in Informationseinheiten aufgegliedert, die sich dann über Referenzen in verschiedene Publikationen einbinden lassen., page 3

4 http://www.tekturcms.de
Das ist die private Homepage des Autors mit einer kompletten Liste seiner Hobby-Projekte seit 2000., page 3

5 https://openclipart.org/
Website von der Open Clipart Bibliothek, page 4

6 https://de.wikipedia.org/wiki/TIOBE-Index
Jährlicher Inder der beliebtesten Programmiersprachen, page 7

7 https://bit.ly/2ARgKCJ
Beleg, dass XSLT einmal im TIOBE Index erschienen ist, page 7

8 https://en.wikipedia.org/wiki/ASC_X12
Wikipedia Seite zum X12 Standard für Electronic Data Interchange, page 8

9 https://en.wikipedia.org/wiki/S1000D
Wikipediaseite zum S1000D Standard, page 8

10 https://de.wikipedia.org/wiki/XSL_Transformation
Wikipedia Seite zur Programmiersprache XSLT, page 8

11 http://www.unidex.com/turing/utm.htm
Turing Maschine in XSLT programmiert, page 8

12 https://de.wikipedia.org/wiki/XSL_Transformation
W3C Seiten zu The Extensible Stylesheet Language Family (XSL), page 8

13 https://de.wikipedia.org/wiki/Document_Object_Model
Wikipedia Seite zum Document Object Model, page 8

14 https://www.w3.org/TR/xsl/
Spezifikation der Auszeichnungssprache XSL-FO für die Formatierung als PDF, page 8

15 https://de.wikipedia.org/wiki/Verarbeitungsanweisung
Die Processing Instruction wertet der Parser als Kommando aus und nicht als Teil des XML Contents, page 9

16 https://de.wikipedia.org/wiki/Wireless_Application_Protocol
Mittels dieser Technologie wurden Webinhalte auf Handys gespielt. Das war vor den Smartphones, page 10

17 https://de.wikipedia.org/wiki/EPUB
EPUB ist ein Dokumentformat für Ebook-Reader., page 11

18 https://de.wikipedia.org/wiki/CHM_(Dateiformat
Die alte Windows-Hilfe. Läuft immer noch im Bereich Maschinenbau auf gekoppelten Rechnern mit alter Windows Software), page 11

19 https://www.ibm.com/developerworks/library/os-echelp/index.html
Das Hilfe-Format der Eclipse Rich Client Plattform. Eclipse wird hauptsächlich von Programmierern als Editor benutzt, page 11

20 https://en.wikipedia.org/wiki/JavaHelp
Damit wird bspw. das Java API formatiert als Webseite ausgegeben, page 11

21 https://de.wikipedia.org/wiki/FrameMaker
Mit Framemaker kann man manuell gesetzte Publikationen erstellen. Über Templates lässt sich das Layout aber auch automatisieren, page 11

22 https://www.i4icm.de/forschungstransfer/pi-mod/
PI-Mod ist ein Informationsmodell, das am KIT (Uni Karlsruhe) entwickelt wird/wurde, page 11

23 https://de.wikipedia.org/wiki/Journal_Article_Tag_Suite
JATS ist ein sehr verbreitetes Informationsmodell im Bereich wissenschaftlicher Artikel und Fachliteratur, page 11

24 https://de.wikipedia.org/wiki/Text_Encoding_Initiative
Wikipediaseite zum TEI Standard, page 11

25 http://surguy.net/articles/client-side-svg.xml
Automatische Image Erzeugung mit XSLT und SVG, page 13

26 http://jackrabbit.apache.org/jcr/node-type-visualization.html
Visualisierung von Knoten im Apache Jack Rabbit Projekt, page 13

27 https://github.com/search?q=xsl%3Astylesheet&type=Code
GitHub Suche nach XSLT Code Schnippseln mit 14Mio Treffern, page 13

28 http://argouml.tigris.org
ArgoUML ist ein freier UML Editor, page 15

29 http://butterflycode.sourceforge.net
Damit kann man sich Code aus UMl Klassendiagrammen generieren lassen, natürlich XSLT basiert, page 15

30 https://www.javaworld.com/article/2073998/java-web-development/generate-javabean-classes-dynamically-with-xslt.html
Weiterführende Lektüre zum Thema Code-Generierung mit XSLT, page 15

31 http://www.saxonica.com/html/documentation/sourcedocs/streaming/
Streaming ist eine Technik zur Verarbeitung großer XML Daten - Stichwort Big Data - mit XSLT3.0, page 15

32 https://de.wikipedia.org/wiki/Hurenkind_und_Schusterjunge
Wikipedia-Seite zum Thema Hurenkinder und Schusterjungen, page 17

33 https://iirds.org/
Webseite zum iiRDS Standard, page 17

34 https://en.wikipedia.org/wiki/Component_content_management_system
Wikipedia Seite zum Thema Component Content Management, page 17

35 http://www.tekturcms.de
Homepage von Tektur CCMS, page 17

36 https://de.wikipedia.org/wiki/XML_Schema
XML Schema ist der Nachfolger der DTD, ist XML basiert und erlaubt auch die die Content-Validierung in einem bestimmten Umfang, page 25

37 http://www.xsltfunctions.com/
Sehr gut gegliederte Funktionsbibliothek von Priscilla Walmsley, page 25

38 https://de.wikipedia.org/wiki/Dokumenttypdefinition
Wikipediaseite zum Thema DTD, page 28

39 https://de.wikipedia.org/wiki/XML_Schema
Wikipedia Seite zu XML Schema, page 28

40 http://www.perfectxml.com/SoftDetails.asp?id=216
Letzte Ressource im Web zum Near & Far Designer DTD Modelling Tool, page 28

41 https://www.gds.eu/de/redaktionssysteme/weitere-loesungen
Webseite zum Tool TreeVision von der Ovidius GmbH, page 28

42 https://www.oxygenxml.com/xml_editor/rng_visual_schema_editor.html
Grafische Komponente des oXygen XML Editors für Schemas, page 28

43 https://de.wikipedia.org/wiki/Backus-Naur-Form
Wikipediaseite zur Backus-Naur Form, page 29

44 https://relaxng.org/tutorial-20011203.html#IDAHDYR
RelaxNG Tutorial, page 30

45 https://relaxng.org/compact-tutorial-20030326.html
Tutorial zur RelaxNG Kurzform, page 30

46 https://de.wikipedia.org/wiki/Backus-Naur-Form
Wikipediaseite zur BNF, page 31

47 https://www.xml.com/pub/a/2004/02/11/relaxtron.html
Artikel zum Einbetten von Schematron in RelaxNG auf xml.com, page 32

48 https://www.w3.org/TR/xml-model/
Associating Schemas with XML documents 1.0 (Third Edition), page 33

49 https://relaxng.org/jclark/jing.html
Homepage des Jing RelaxNG Schema Validators, page 35

50 http://www.jclark.com/bio.htm
Homepage con James Clark, page 35

51 http://www.jclark.com/sp/
Homepage zum SP SGML Parser, page 35

52 https://relaxng.org/jclark/trang.html
Multi-Schema Konverter Trang, page 36

53 https://de.wikipedia.org/wiki/XML_Schema
Wikipediaseite zum XML Schema, page 36

54 https://github.com/citation-style-language/utilities/blob/master/RNG2Schtrn.xsl
Rohes Skript zum Extrahieren von Schematron Regeln aus einer RNC Kompaktform, page 37

55 https://de.wikipedia.org/wiki/Darwin_Information_Typing_Architecture
Wikipedia Seite zum DITA Standard, page 37

56 https://de.wikipedia.org/wiki/DocBook
Wikipedia Seite zum Docbook Standard, page 37

57 https://wiki.selfhtml.org/wiki/XML/DTD/Attribute_und_Wertzuweisungen
Erläuterungen zu Attributen und Wertzuweisungen auf selfhtml.org, page 39

58 https://www.ibm.com/developerworks/library/x-dita2/index.html
Artikel (eng.) zum Thema DITA Spezialisierung auf einer IBM Website, page 39

59 https://github.com/dita-ot/dita-ot
GitHub Seite zum DITA Open Toolkit, page 40

60 https://docs.oasis-open.org/dita/v1.0/langspec/ul.html
DITA Doku zum <ul> Element

61 https://de.wikipedia.org/wiki/XSL_Transformation#MetaMorphosis
Wikipedia Eintrag zu dem XML Prozessor Metamorphosis, page 49

62 http://erlang.org/doc/man/xmerl_xs.html
Dokuseiten zum Modul xmel_xs, mit dem man in Erlang wie in XSLT "programmieren" kann, page 50

63 https://de.wikipedia.org/wiki/Elektronischer_Datenaustausch
Wikipediaseite zum Elekronischen Datenaustausch / EDI, page 54

64 https://de.wikipedia.org/wiki/Prozessmanagement
Wikipediaseite zum Prozessmanagement, page 55

65 https://camunda.com/
Website der Camunda BPMN Engine, page 55

66 https://github.com/alexdd/tektur_worker
Github Repo eines Camunda Task-Executors für XML Transformationen, page 55

67 https://en.wikipedia.org/wiki/Template_method_pattern
Wikipediaseite zum Template Method Pattern, page 59

68 https://docs.python.org/2/library/sgmllib.html
Einfacher SGML Parser der Python Standard Bibliothek, page 60

69 https://www.saxonica.com/html/documentation/sourcedocs/streaming/xslt-streaming.html
XSLT3.0 Streaming API, page 60

70 https://www.saxonica.com/html/documentation/sourcedocs/streaming/
page 60

71 https://www.saxonica.com/html/documentation/xsl-elements/mode.html
Mode Optionen in XSLT3.0, page 61

72 https://www.saxonica.com/html/documentation/xsl-elements/iterate.html
Der Iterator ist ein Konzept um XSLT Streaming zu realisieren, page 63

73 https://www.mongodb.com/
Homepage der NoSQL Datenbank MongoDB, page 77

74 http://www.jclark.com/xml/xmlns.htm
Webpage von James Clark bezüglich XML Namespaces, page 79

75 http://www.xsltfunctions.com/xsl/functx_trim.html
FunctX Bibliothek von Priscilla Walmsley zum Thema Trimming, page 79

76 https://www.w3.org/TR/DOM-Level-3-Core/core.html#Document3-normalizeDocument
Spezifikation zum DOM 3 Core auf den Seiten des W3 Konsortiums, page 80

77 http://www.xsltfunctions.com/xsl/functx_replace-multi.html
Die Multi-Replace XPath Funktion von Priscilla Walmsley auf der Website der xsltfunctions Bibliothek, page 85

78 https://de.wikipedia.org/wiki/%C3%85
Wikipediaseite zum Bolle-A, page 85

79 http://www.xsltfunctions.com/xsl/c0015.html#c0052
FunctX Bibliothek von Priscilla Wamsley - Hier die Sektion zum Thema Comparing, page 86

80 https://www.html-tidy.org/
Website zu HTML Tidy Markup Corrector, page 87

81 https://www.codetable.net/decimal/148'Webreferenz zum Entity 148
page 87

82 https://www.saxonica.com/html/documentation9.8/extensions/output-extras/serialization-parameters.html
Dokuseite zu den Saxon Serialization Parametern, page 88

83 http://www.unicode.org/Public/MAPPINGS/VENDORS/ADOBE/symbol.txt
Unicode Mapping Tabelle für den Symbol Font, page 88

84 Unicode Mapping Tabelle für den Zapf Dingbats Font
page 88

85 https://web.archive.org/web/20060924061009/http://ppewww.ph.gla.ac.uk/~flavell/charset/fontface-harmful.html
Seite im Internet Archive, die die Probleme die Firefox mit dem Font "Symbol" hat gut beschreibt, page 88

86 https://lxml.de/
Homepage der lxml Python library, page 89

87 https://gitlab.gnome.org/GNOME/libxslt
Homepage der C Bibliothek libxslt, page 89

88 https://exslt.github.io/
Github Seite der EXSLT Erweiterungen, page 89

89 https://github.com/alexdd/bandit-sarif/blob/feature-sarif-formatter-with-cwe-extension/bandit/formatters/sarif/sarif.xsl
Github Repo zum Bandit Sarif Report, page 91

90 http://exist-db.org/exist/apps/homepage/index.html
Hompage der eXist XML Datenbank, page 91

91 https://de.marklogic.com/
Hompage der NoSQL/XML Datenbank MarkLogic, page 91

92 https://www.oxygenxml.com/xml_editor/xquery_builder.html
Tool zur einfachen Eingabe von XQuery Test-Skripten im oXygen XML Editor, page 91

93 http://cs.au.dk/~amoeller/XML/querying/flwrexp.html
XQuery Extensions für mächtigere Funktionen, page 92

94 https://www.oxygenxml.com/xml_editor
Homepage zum oXygen XML Editor, page 92

95 http://www.saxonica.com/documentation/#!sourcedocs/projection
Verstecktes Saxon Feature: Dokument Projektion bei einer XQuery Abfrage, page 93

96 https://docs.marklogic.com/xdmp:document-insert
Doku zu xdmp:documen-insert Funktion auf den MarkLogic Webseiten, page 94

97 https://docs.marklogic.com/xdmp:document-add-collections
Dokumentation zur xdmp:document-add-collections Funktion auf den MarkLogic Webseiten, page 95

98 https://docs.marklogic.com/guide/app-dev/TDE
Template Driven Extraction wird verwendet um in MarkLogic eine relationale Sicht auf die baumstrukturierten Daten zu setzen, page 97

99 https://developer.marklogic.com/recipe/move-a-document/
Rezept, wie man ein Dokument in MarkLogic "umbenennt"

[EX] http://exist-db.org/exist/apps/homepage/index.html
Homepage zur eXist XML Datenbank

[BX] http://basex.org/
Homepage zur BaseX XML Datenbank

[ML] https://www.marklogic.com/
Homepage zur XML Datenbank MarkLogic

[BD] https://www.oracle.com/database/berkeley-db/xml.html
Website zur Berkeley DB XML Datenbank auf den Seiten von Oracle

[DL] https://developer.marklogic.com/free-developer
MarkLogic Developer Lizenzvereinbarung, page 104

105 https://www.oxygenxml.com/doc/versions/20.1/ug-editor/topics/configure-marklogic-datasource.html
MarkLogic Datenquelle in oXygen konfigurieren, page 105

106 https://en.wikipedia.org/wiki/Temporal_database
Wikipedia Eintrag zum Thema Temporal Databases mit einer Begriffserklärung, page 110

107 https://docs.marklogic.com/guide/temporal
Quickstart Dokumentation, Understanding, Managing and Searching Temporal Documents, page 112

108 https://www.marklogic.com/blog/bitemporal/
Weiterführender Link zum Thema Bitemporale Dokumente in MarkLogic, page 112

109 https://www.heise.de/developer/artikel/Temporale-Datenhaltung-in-der-Praxis-mit-Java-2100268.html?seite=all
Temporale Datenhaltung in der Praxis mit Java, page 113

110 https://de.wikipedia.org/wiki/Anonyme_Funktion
Wikipedia Artikel zum Begriff Anonyme Funktion, page 115

111 https://docs.marklogic.com/guide/app-dev/transactions
Arbeiten mit Transaktionen in MarkLogic Server, page 115

112 https://docs.marklogic.com/guide/temporal/searching#id_92200
ISO normierte Operatoren in MarkLogic zum Vergleichen von Zeiträumen, page 120

113 https://docs.marklogic.com/guide/temporal/searching#id_98704
Allen (ALN) Vergleichsoperatoren für Zeiträume, page 120

114 https://de.wikipedia.org/wiki/CURL
Wikipedia-Seite zu Curl URL Request Library - cURL, page 120

115 https://docs.marklogic.com/REST/POST/manage/v2/databases
Doku-Seite zum Anlegen einer MarkLogic Datenbank mit cURL, page 121

116 https://daniel.haxx.se/blog/2016/08/19/removing-the-powershell-curl-alias/
Anleitung zur Entfernung des curl Alias auf Windows, page 121

117 https://docs.marklogic.com/guide/admin-api/configure
XQuery Skripte zur Konfiguration von MarkLogic, page 123

118 https://docs.marklogic.com/xdmp:eval-in
Eval-in Funktion auf der MarkLogic Dokuseite, page 127

119 https://sourceforge.net/p/saxon/mailman/message/13252035/
Eine Diskussion zum Thema URI Resolver in Saxon mit Michael Kay, page 130

120 https://dumps.wikimedia.org/enwiki/latest/
Listing aller XML Dumps von Wikipedia, page 131

121 https://github.com/fgeorges/expath-ml-console
ML EXPath Konsole auf GitHub, page 135

122 http://mlproj.org/
Project and Environment Manager für MarkLogic, page 138

123 https://github.com/marklogic/marklogic-contentpump
GitHub Projekt zur MarkLogic Content Pump, page 138

124 https://developer.marklogic.com/products/mlcp
Binaries zur Installation der MarkLogic Content Pump, page 138

125 https://docs.marklogic.com/guide/mlcp/import
Dokuseite zu den Import Optionen der MarkLogic Content Pump, page 138

126 https://github.com/marklogic-community/ml-gradle
Projektseite auf GitHub zu ml-gradle, page 141

127 http://mlproj.org/
Website des mlproj Projekts, page 141

128 https://gradle.org/
Website des Build-Tools gradle, page 141

129 https://github.com/marklogic-community/ml-gradle
Projektseite auf GitHub zu ml-gradle, page 142

130 https://docs.marklogic.com/guide/rest-dev
Dokuseite zur MarkLogic REST API, page 142

131 https://github.com/marklogic-community/ml-gradle/wiki/Property-reference
Properties für das MarkLogic Deployment-Tool ml-gradle, page 142

132 https://github.com/marklogic-community/ml-gradle/wiki/Generating-new-resources
Dokuseite zu ml-gradle bzgl. der Tasks zum Erzeugen neuer Ressourcen, page 142

133 https://github.com/marklogic-community/ml-gradle/wiki/Configuring-resources
Dokuseite zu ml-gradle bzgl. Konfiguration von "Payload-Files", page 142

134 https://github.com/marklogic-community/ml-gradle/wiki/Debugging-module-loading
Beispiel für einen einfachen Groovy-Task auf der Dokuseite zu ml-gradle, page 142

135 http://fgeorges.org/
Homepage von Florent Georges, page 142

136 https://www.antennahouse.com/formatter/
Homepage der AntennaHouse Formatter Software zur Formatierung von PDF Dokumenten, page 143

137 https://xmlgraphics.apache.org/fop/
Homepage des Apache Formatting Objects (FOP) Projekts, page 143

138 https://en.wikipedia.org/wiki/Printer_Command_Language
Wikipedia Seite zur Printer Command Language von HP, page 143

139 https://www.w3.org/TR/xsl/
Spezifikation zur Auszeichnungssprache XSL-FO, page 143

140 https://www.w3.org/TR/2010/REC-xpath-functions-20101214/
XPath Funktionskatalog auf den Seiten des W3 Konsortiums, page 143

141 http://www.tekturcms.de/stylesheets/margits_noten.pdf
XSL-FO Experiment, das eine Notentabelle für die Kollegstufe am Gymnasium generiert, page 144

142 https://de.wikipedia.org/wiki/Endrekursion
Wikipedia Seite zum Thema Endrekursion, page 147

143 http://www.heise.de
kurze beschreibung, page 149

144 https://github.com/Schematron/schematron
Schematron auf GitHub, page 150

145 https://github.com/xspec
XSpec auf GitHub, page 152

146 https://docs.oracle.com/javase/8/docs/technotes/guides/troubleshoot/memleaks002.html
Doku-Seite auf Oracle zum Thema Memory Fehlermeldungen, page 154

147 https://www.oracle.com/technetwork/java/gc-tuning-5-138395.html
Doku-Seite auf Oracle zum Thema Heap-Memory Einstellungen und Garbage Collector, page 154

148 https://www.xml-project.com/morganaxproc/
Homepage zum MorganaXProc Projekt, page 154

[XM] https://xmetal.com/
Homepage des Desktop XML Editors XMetal

[EP] https://www.ptc.com/en/products/service-lifecycle-management/arbortext/editor
Website zum Arbortext XML Editor

[OX] https://www.oxygenxml.com/oxygen-xml-web-author/app/oxygen.html
Der oXygen XML Web Editor im Web

[FX] https://www.fontoxml.com/
FontoXML Webeditor

[XE] http://www.xeditor.com/portal
Homepage des webbasierten XEditors

[XO] http://xopusfiddle.net/VT7T/3/
Homepage des veralteten Xopus XML Editors

155 https://de.wikipedia.org/wiki/Revision_Control_System
Wikipediaeintrag zu RCS, page 157

156 https://de.wikipedia.org/wiki/Concurrent_Versions_System
Wikipediaeintrag zu CVS, page 157

157 https://www.cvedetails.com/vulnerability-list/vendor_id-442/CVS.html
Eintrag zu CVS in der „Common Vulnerabilities and Exposure“ Datenbank, page 158

158 https://de.wikipedia.org/wiki/Apache_Subversion
Wikipediaeintrag zu Subversion, page 158

159 https://de.wikipedia.org/wiki/Git
Wikipedia Eintrag zu GIT, page 158

160 https://de.wikipedia.org/wiki/Git
Wikipedia Eintrag zu GIT, page 158

Index

Appendix

Q&A 2022 / 09	XML Hands-On XSLT & co.				Company	X
					Scope	X
					Capacity	X
					Terms	X
	Date:	2022/09/11	**Status:**	DRAFT	**Features**	X

Tektur CCMS Commercial Status

Topic:	Just another test document	**Revision:**	1
Author:	Alex Düsel	XML CCMS Developer	

Market Sounding Questionaire

Just recently I was asked to complete a Market Sounding Questionnaire regarding DITA CCMS systems. Although Tektur CCMS is a side project of mine and still under development I will take the opportunity to give you the requested information in the form of a quick "Memo" test document, that was created just by copying and pasting the Word document that you sent me into the browser-based editor. Please note that the real questionaire contains a lot more questions. I just picked a few.

- Registered company name: Tektur CCMS
- Company registration number / Tax No: DE333673306
- Country of company registration: Germany
- Key contact person name: Alex Düsel
- Key contact person email: tekturcms@gmail.com
- Key contact person phone, including country dialling code if not UK: ++49 163 312 14 12
- How long has your organisation been in the CCMS market?
  I have been working in the CCMS market for more than 15 years mainly as an XML Developer

CCMS pilot project scope:

A. Preparation and maintenance of specifications for:		
A.1	The pilot project DITA CCMS	X
B. Planning, implementation and maintenance of a pilot project CCMS for circa 20 users, including but not limited to:		
B.1	Programme (project plan)	X
B.2	Information model	X
B.3	Metadata schema	X
B.4	Reuse strategy	X
B.5	Workflow specification	X
B.6	Authoring guidelines	X

B.7	Style sheets	X
B.8	Training plan	X
C. Policies, processes and procedures for		
C.1	Content conversion	X
C.2	Version management	X
C.3	Archiving	X
C.4	Deletion	X
C.5	Style sheets	X
D. Procurement, installation, configuration and maintenance of the pilot project CCMS X		
E. Training of all pilot project CCMS users, including but not limited to authors, reviewers, approvers and system administrators. X		

- Can your organisation deliver all parts of the CCMS pilot project scope? If not, which parts would your organisation be unable to deliver? YES
- Which parts of the CCMS pilot project scope would your organisation sub-contract, if any? Depending on the quality of the PDF output we may consider Antennahouse PDF formatter (commercial third party product) instead of Apache FOP PDF formatter (open source solution)
- Would your organisation supply all necessary software licences for the CCMS pilot project or would these need to be procured from a third party? If the latter, please list available suppliers of the necessary licences. YES
- Does your organisation have capacity to start delivering the CCMS pilot project in October 2022? If not, by when could your organisation start delivery? NO (unfortunately I have just started a full-time job that does not allow me to have any commercial side work)
- Agile delivery / While XXX has developed a draft specification and user stories, there is some uncertainty regarding the following:
  - Prioritisation of requirements (Must have, Should have, Could have, Won't have, MoSCoW)
  - Number of existing documents to be converted
  - Interfaces with other systems
  - Final scope of the CCMS pilot project
- Please specify which agile project delivery frameworks are used by your organisation and the duration of your organisation's experience in delivering projects using each agile framework. Tektur CCMS has currently been developed as a One-Man-Project with minimum project management. In my daywork we typically use a SCRUM based process.
- Contract terms (NOTE: A XXX-wide CCMS is not currently budgeted and would be subject to the outcomes of the CCMS pilot project.)
  - For the pilot project, XXX is proposing an initial contract term of 12 months with an option to extend by an additional 12 months. Please provide any feedback on the proposed term.

- Contract value (NOTE: A XXX-wide CCMS is not currently budgeted and would be subject to the outcomes of the CCMS pilot project.)

- What are the indicative costs for the following scope items?
  - Preparation and maintenance of the CCMS specifications 80000 EUR
  - Planning and implementation of a pilot project CCMS for circa 20 users 120000 EUR
  - Cost per user per month to maintain the pilot project CCMS 100 EUR / User

- Please complete the table to detail the CCMS software that your organisation can supply and their conformity to the principal requirements in Annex 1: Draft CCMS specification.

CCMS product features

Product manufacturer	Tektur CCMS
Current version	0.1 / prototype
Update frequency	once a month
Conforms to BS ISO/IEC/IEEE 26531	No
DITA	The internal browser-based editor supports a (large) subset of the DITA model. It is intentend to use oXygen XML desktop based editor for advanced editing.
Cloud service with 99.9% uptime that meets the CIF code of practice	Cloud support will be added in the future. Currently the system runs on-premise.
Web user interface	YES, Full web interface
MS Office user interface	Word and Excel Content can be integrated just by copy 'n paste
Interoperable Please specify any application programming interfaces (APIs).	There is a minimal REST API for managing DITA topics
Publish PDF/A, EPUB and HTML 5	No / EPUB but PDF, HTML package (offline support), online HTML5 output, DITA XML, CHM
Import and export ReqIF	No
Comments	System is still under initial development

- Experience
  - Please detail your organisation's experience in delivering similar scopes to meet the needs and expectations of your customers. Supporting documented information may be submitted, e.g. case studies, customer testimonials.
    - Preparation and maintenance of CCMS specifications Worked in a technical project manager role in the German tec-doc industry for about 2 years
    - Planning and implementation of a pilot project CCMS for circa 20 users Used to work as a XML developer with two major CCMS companys / more than 20 years experience in the market

- Maintenance of a pilot project CCMS dito

- Please detail any other relevant experience that your organisation has. Software Developer with more than 20 years of experience

- Please detail any relevant innovations that are likely to emerge during the course of the pilot project (12 to 24 months from September 2022). This pilot project could be also used to put the final touch onto the browser-based XML editor that feels like M$ Word but allows for the input of complex semantic XML structures :
  - Information on the development of microcontent NO
  - Mathematics and vector graphics MAYBE
  - Classification of objects using metadata and taxonomies YES
  - Webhooks and triggers NO
  - XML reviews using Schematron or other similar systems MAYBE
  - Reporting capabilities YES
  - Dynamic content generation YES

- Risks and opportunities. Please describe the top three risks to the successful delivery of the scope.

Rank	Risk and description	Mitigating actions	Who is best placed to manage this risk (XXX or Supplier)
1	Content Conversion runs into problems due to corrupt XML / other structures	Filter corrupt structures / Focus on relevant data	both
2	Current UI / UX of Tektur CCMS is not accepted by Users	Better understanding of underlying concepts	XXX
3	PDF quality is not accepted	Change to commercial PDF formatter	both

- Please describe the top three opportunities for exceeding the successful delivery of the scope.

Rank	Opportunity and description	Enabling actions	Who is best placed to manage this opportunity (XXX or Supplier)
1	Tektur CCMS source code and features improved and put as Open Source	Spend enough time on code quality	Supplier

2	Multiple users allow for evaluating real time features such as collaborative editing	Good Key User feedback, support and willing to try out new features	both
3	Participate in existing Open Source projects such as the DITA OTP and benefit from the community work	XXX is also willing to engage and sponsor open source development	both

Company Info

Tektur CCMS is a side-project of IT Consulting Alex Düsel (http://www.tekturcms.de)
Please send your requests to tekturcms@gmail.com

Roadmap 2021 - 25	XML Hands-On XSLT & co.				Node Server	X
					JS	X
					XSLT	X
					Django	X
	Erstellt:	05.04.2021	**Freigabe:**	01.12.2021	MongoDB	X

e-Learning mit Tektur *LCMS*

Status:	DRAFT	**Revision:**	1
Autor:	Alex Düsel	XML- / CCMS-Entwickler	

Motivation

Zur *Technischen Dokumentation* gibt es in den Branchen Luftfahrt, Maschinenbau und Automotive spezielle Software, die eine *topicbasierte*, *teilautomatisierte* und *wiederverwendbare* Dokumentation von Bauteilen ermöglicht.

Dabei können Betriebsanleitungen mit mehreren Tausend Seiten entstehen, die für verschiedene Ausgabestrecken automatisch gesetzt werden. Diese sog. *Component Content Management Systeme (CCMS)* sind mit sehr hohen Einführungs- und Betriebskosten verbunden und werden seit mehr als 20 Jahren als Desktopsoftware mit Serverkomponente für Windows OS entwickelt.

Mit aktueller Cloud- / Webtechnologie und Open-Source Software ist dieser Anwendungsbereich aber mittlerweile auch komplett webbasiert abbildbar. Es gibt jedoch weltweit noch sehr wenige Anbieter, die konsequent webbasiert arbeiten.

Der *CCMS* Markt ist in Deutschland zwischen einer Handvoll größerer Anbietern aufgeteilt, die mittels spezialisierter Inhaltsmodelle ihre Kunden seit Jahrzehnten an sich gebunden haben. Deren Systeme sind aber teilweise technologisch veraltet, so dass mittelfristig eine Neuorientierung stattfinden wird.

Es würde den Rahmen von *Tektur CMS* sprengen einen vollständigen Ersatz für die existierende Software zu bieten, da in der Technischen Dokumentation eine Vielzahl unterstützender Software verwendet wird, wie z.B. für das Terminologiemanagement, Übersetzungsmanagement, Bildbearbeitung, Workflowsysteme für Review- und Freigabezyklen, Content Delivery Portale, etc. Diese Schnittstellen müssten ebenfalls bedient werden.

Was aber möglich wäre, und was in der aktuellen Zeit sicherlich für viele Unternehmen einen Mehrwert darstellen würde, wäre sich auf den Bereich e *-Learnings* als führendes Ausgabeformat mit einer *Learning CMS Software (LCMS)* zu spezialisieren.

Konzept und Technologische Orientierung

- Interaktive *e-Learning* Module stellen in der heutigen Arbeitswelt mit Homeoffice und *Lifelong Learning* einen essentiellen Bestandteil im Wissensmanagement in der Industrie dar. Auch im Bereich *Edutainment* und nicht zuletzt im universitären oder Schulbetrieb wären automatisierte Prozesse, mit denen man schnell und unkompliziert interaktive Lernmodule mit definierten Schnittstellen zu anderen Systemen erstellen könnte, vorteilhaft.
- Dabei stellt die Wiederverwendbarkeit von möglichst feingranular zerlegten und einmal übersetzten Lerninhalten eine zentrale Herausforderung dar.

- Hier kann man sich die Konzepte aus der Technischen Dokumentation zu Nutze machen. Dabei wird mittels wiederverwendbarer *XML Bausteine*, die nach dem *Single-Sourcing* Prinzip referenziert und nicht kopiert werden, in verschiedenene Ausgabeformate publiziert.
- Im Bereich *e-Learning* wäre das führende Ausgabeformat ein *autonomes, Interaktives HTML5 Format*, das vom Lernenden heruntergeladen werden kann, oder aber auch Online mit Zusatzfunktionalität ausgeführt werden kann.
- Zusätzlich sollte es noch ein herkömmliches PDF Format geben, welches die Lerninhalte ohne Interaktion zur Archivierung aufbereitet und verschiedene Statistiken zur späteren Auswertung bereithält.
- Die Quelldaten würden dabei medienneutral und zukunftssicher in einem gängigen XML Format, wie z.B. *DITA (Darwin Information Typing Architecture)*, verwaltet.
- Um die Erstellung der Lerninhalte einem *Content Management* und einem kollaborativen *Prüf- und Freigabezyklus* zu unterziehen, sollten die Lerninhalte über eine *cloudbasierte Webapplikation* eingepflegt werden können, die auch die gängigen Funktionen eines *CCMS Systems*, wie *Versionierung*, *Modularisierung*, *Variantensteuerung*, *Rechte- und Rollenmanagement*, *Workflow*, etc. bereitstellt.

Aktueller Bestand

Zum 05.04.2021 kann man mit *Tektur CMS* Inhalte nach dem *DITA* Inhaltsmodell webbasiert erfassen, und diese mittels gängiger Portalfunktionalitäten in verschiedenen Ausgabeformaten (HTML5, PDF, RTF, DITA), Papierformaten (A4, A5) und Layouts konfigurierbar auf einer Website publizieren.

Zusätzlich besteht die Möglichkeit einzelne Topics einem Prüf- und Freigabezyklus zu unterziehen. Dabei können Lektoren und Prüfer Diskussionen an einzelnen Textpassagen anbringen und diese schließlich freigeben, so dass der verantwortliche Redakteur abgestimmte Änderungen an der Publikation vornehmen kann.

Features 2022

Im Jahr 2022 soll *Tektur CMS* zu einem DITA System für Gelegenheitsredakteure ausgebaut werden. Wichtige noch fehlende Funktionalitäten sollen ergänzt werden und bestehende geschliffen werden, so dass Ende 2022 eine kleine, aber feine Lösung vorhanden ist.

Nr	Titel	Beschreibung	Notizen	Erledigt
202201	Weitere Importformate AsciiDoc und ein Exportformat eines kommerziellen CCMS.	Zusätzlich zu den bereits vorhanden Formaten Markdown, Word und Tektur DITA sollen noch weitere Formate importiert werden können.	Das Format AsciiDoc kann über das Opensource Tool AsciiDoctor realisiert werden. Das kommerzielle Exportformat kann natürlich nur nach Absprache mit dem Hersteller importiert werden.	☐
202202	Weitere Ausgabeformate CHM, EclipseHelp, Markdown, Normalized DITA	Neben den existierenden Formaten PDF, HTML5, HTML Paket und DITA XML sollen noch unkompliziert weitere Formate angeboten werden.	Die weiteren Formate sollen über das DITA Open Toolkit integriert werden. Ggf. können auch die OT Formate PDF und HTML eingebunden werden, sollen aber die Tektur eigenen HTML und PDF Formate nicht verdrängen.	☐

Nr	Titel	Beschreibung	Notizen	Erledigt
202203	Diffing mit Vorgängerevisionen	Derzeit können Vorgängerrevisionen eines Topics, Tasks, Memos oder einer Map nur angezeigt werden, aber nicht verglichen werden.	Da in Tektur jedes Element mit einer eindeutigen ID ausgezeichnet ist, kann ein einfacher Diffing Algorithmus angewendet werden, der ausgehend von der ID geänderte, neu hinzugekommene oder gelöschte Texte und Grafiken markieren kann.	☐
202204	Night Mode im HTML5 Format	Im öffentlichen Frontend des Portals soll es einen Umschalter für einen augenschonenden "Nachteulen"-Modus geben.	Der Nachtmodus soll mittels CSS Variablen umgesetzt werden. Dabei soll auch das CSS optimiert werden.	☐

Features 2023

Im Jahr 2023 soll *Tektur CMS* weiter geschliffen werden. Insbesondere soll es weitere Konfigurationsmöglichkeiten bzgl. der Ausgabeformate geben. Dazu wird der vorhandene Layouter-Dialog weiter ausgebaut und auch die Möglichkeit zum Online Bearbeiten einer umfangreichen Konfigurationsdatei gegeben werden.

Nr	Titel	Beschreibung	Notizen	Erledigt
202301	Element- und Strukturgültigkeiten	Im Kapitelstruktureditor sollen über ein Dropdown zuvor definierte Gültigkeiten an den Topics und Tasks gesetzt werden können. Im DITA Content sollen Gültigkeiten über das Lazy-Tag Feature funktionieren.	Die Gültigkeiten sollen in einem Reiter im Map-Dialog definiert werden können.	☐
202302	"Wussten Sie schon?"-Dialog beim Applikationsstart	Beim Start der Webanwendung soll ein optionaler Dialog angezeigt werden, der jeden Tag einen neuen Tipp vorschlägt. Dazu müssen die Tipps gesammelt werden, ggf. mit dem Memo-Editor.	Der bestehende "Willkommen" Dialog soll um diese Funktion erweitert werden.	☐

Nr	Titel	Beschreibung	Notizen	Erledigt
202304	Sprechende Dateinamen für die Export Formate.	Derzeit sind die Dateinamen der Exportformate willkürlich bezeichnet, bspw. "result.pdf" für das PDF Format. Der Dateiname soll den Titel des Informationsobjekts enthalten, sowie die Revisionsnummer und das Exportdatum / Zeit.	--	☐
202305	Erweiterte Konfigurationsmöglichkeit für die Ausgabeformate.	Neben dem bestehenden Layouterdialog soll es auch ein Konfigurationsfile geben, das mittels XML strukturiert ist und online bearbetet werden kann.	Die Konfigurationsdatei soll global für alle Formate der Tekturinstallation greifen. Die Funktionalität ist nicht pro User oder pro Informationsobjekt vorgesehen.	☐
202306	Übersetzungssteuerung einbauen	Momentan muss man jede Sprache für jede Map, Topic, Task oder Memo Objekte separat anlegen. Das soll automatisiert werden.	Die "translate" Funktionen werden analog zu den "review" Funktionen eingebaut. Wenn ein Objekt im Workflow "translate" ist, dann erscheint es beim ausgwälhlten Translator in einem separaten Listing und ist für alle anderen Funktionen gesperrt.	☐

Features 2024

Im Jahr 2024 soll *Tektur CMS* in die Cloud und an einem Subscription Preis-Modell gearbeitet werden - Stichwort Software-as-a-Service.

Nr	Titel	Beschreibung	Notizen	Erledigt
202401	Lokale Cloud-Umgebung aufsetzen	Um ordentlich Entwickeln, Testen und Debuggen zu können, ist es notwendig auf lokalen Rechnern eine Cloud aufzusetzen, bevor man produktiv geht.	Das AWS Localstack Projekt soll hierzu verwendet werden.	☐
202402	Tektur-Instanzen per Admin-Interface anlegen und verwalten.	Ein Administrator kann neue Tektur-Instanzen in der Cloud komfortabel anlegen und verwalten.	Das Admin-interface von *Tektur CCMS* ist mittels Python/Django realisiert, während die Basisapplikation mittels NodeJS realisiert ist. Hier sollen weitere Node-Knoten per Django initialisiert, eingeschaltet, ausgeschaltet und gelöscht werden können.	☐
202403	Skalierungs- und Performanzoptionen in NodeJS	NodeJS ist für ein SaaS-Modell prädestiniert. Hier gilt es die verschiedenen Optionen zu erforschen und anzuwenden, so dass das System optimal läuft.	--	☐
202404	Realtime Features	NodeJS kann Nachrichten an die angeschlossenen Browser pushen. Das ermöglicht eine Vielzahl von Spezialfeatures, die mit herkömmlicher Webentwicklung nicht so einfach umzusetzen sind.	Je nach zeitlicher Kapazität sind die folgenden Funktionalitäten vorstellbar: – Schreib-Lock für Informationsobjekte in Echtzeit aktualisieren - für eine bessere kollaborative Zusammenarbeit der Redakteure – Kommentare und DIskussionen in Echtzeit einblenden – Chats pro Dokument und/oder pro Arbeitsgruppe	☐
202405	Protokollierungs und Mailingfunktionen	Useraktionen sollen umfangreich protokolliert werden und Benachrichtigungsemails sollen verschickt werden.	Die zugehörigen Informationen sollen per Django Admin-Interface verwaltet werden.	☐

Features 2025

Im Jahr 2025 sollen die Arbeiten an der ersten Version von *Tektur CMS* abgeschlossen werden und der Fokus auf das Learning CMS gesetzt werden. Da man davon ausgehen kann, dass in 5 Jahren die Browsertechnologie weiter fortgeschritten ist, soll insbesondere auch nochmal das Thema Browserkompatibilität des XML Editors aufgerollt werden.

Nr	Titel	Beschreibung	Notizen	Erledigt
202501	Learning Schema entwickeln	Um die Eingabe der Elemente einer Learning-Anwendung eingeben zu können, muss zuerst ein Schema erarbeitet werden.	Als Grundlage für das Schema soll der Use Case "Ich möchte interaktiv XML Programmierung Lernen" herangezogen werden. Dabei sollen die bereits bestehenden Testpublikationen auf **www.stylesheet-entwicklung** weiter ausgebaut und professionalisiert werden.	☐
202502	Memo-Editor zum Learning-Editor ausbauen	Das Learning-Schema soll in den Memo Editor möglichst generisch integriert werden	--	☐
202503	XML Editor browserkompatibel machen	Momentan funktioniert der DITA XML Editor nur mit CHROME und Edge. Der DITA XML Editor soll auch fit für den Firefox gemacht werden.	--	☐
202504	Grafikverwaltung	Es soll eine bestehende Open Source Grafik-Verwaltung eingebaut werden.	Hierzu gibt es bestehende Python/Django Apllikationen, die man integrieren kann.	☐
202505	Learning Ausgabeformat	Als Grundlage für das Learning Ausgabeformat soll das bestehende HTML Paket genommen werden, welches um interaktive Learning-Inhalte ergänzt wird.	Es wird unterschieden, zwischen Inhalten, die eine Online-Interaktion benötigen und interaktiven Inhalten, die auch lokal ausgeführt werden können. Die Online-Interaktion sollte nur optional hinzugeschaltet werden, sobald eine Internetverbindung des Users vorliegt.	☐

Geschäftliches

Tektur CCMS ist ein Freizeitprojekt von IT Beratung Alex Düsel (http://www.publiziere.de)
Anregungen und Fragen senden Sie bitte gerne an tekturcms@gmail.com

Tekur CCMS ist ein web-basiertes Component Content Management System und befindet sich noch in der Entwicklung.

Hier sind einige Random Features:

- Die Inhalte werden nach dem DITA Content Model eingegeben. Die Ausgabe erfolgt über ein automatisches Satzsystem.
- Grafiken können für die PDF-Ausgabe seitenbreit, spaltenbreit und in der Marginalie gesetzt werden.
- Die Breite der Marginalie ist stufenlos einstellbar; die PDF-Ausgabe ist bzgl. der Formatierung weitestgehend konfigurierbar.
- Layoutoptionen bzgl. Papierformat, Bemassung und Schriftgrößen können über einen einfachen Dialog eingestellt werden.
- TOC und mehrstufige Register werden automatisch in der PDF-Ausgabe erzeugt.
- Die Zellenbreite von CALS Tabellen kann mit der Maus eingestellt werden; Funktionen auf Zellen sind weitestgehend implementiert.
- Paras, Listitems und Sections können mit den Pfeilbuttons in der Toolbar nach oben und unten verschoben werden.
- Verlinkung auf andere Topics funktioniert über Referenzen und ein Linktext wird automatisch aktualisiert, wenn sich der Topic-Titel ändert.
- Die DITA-Map kann u.a. mittels Drag 'n Drop editiert werden; Im Topic Editor gibt es an jeder Stelle ein dynamisches Kontextmenü für weitere Optionen.
- Valide DITA Strukturen können exportiert und importiert werden.
- Topics, Tasks und Maps können vom Autor an Reviewer und Approver für einen Kommentar- und Freigabeprozess überwiesen werden.