

Laufzeit von verschiedenen XML Strukturen/XPath Ausdrücken am Beispiel einer Autodatenbank

Jost Markus, 3856853

Universität Bielefeld

Informationsstrukturierung (S) (WiSe 2020/2021)

31. August 2021

Inhaltsverzeichnis

1	Einleitung	1
2	Datengrundlage	1
3	Modellierungsentscheidungen	1
4	XPath Abfragen	3
4.1	XPath Abfrage 1	4
4.2	XPath Abfrage 2	4
4.3	XPath Abfrage 3	5
4.4	XPath Abfrage 4	5
4.5	XPath Abfrage 5	5
4.6	Sonstige XPath Abfragen	6
5	Visualisierung der Ergebnisse	7
6	Modellierungen im Überblick	7
7	Fazit	7
8	Ausblick	8

1 Einleitung

Eine der Eigenschaften der Auszeichnungssprache *Extensible Markup Language* (XML) ist die Möglichkeit, Informationen auf mehrere Arten und Weisen modellieren zu können, da man vor allem die Wahl zwischen Elementen und Attributen hat. Dabei stellt sich die Frage, inwiefern sich diese Strukturen in der Verarbeitungszeit unterscheiden. In diesem Projekt wird mithilfe von XML eine Datenbank ausgezeichnet, die über 32.000 Autos mit jeweils 115 Spezifikationen enthält. Die Datenbank wird auf zwei Weisen modelliert und mithilfe von der *XML Path Language* (XPath) abgefragt. Die Laufzeit von Abfragen, die dasselbe Ergebnis liefern, aber einer anderen Struktur unterliegen, wird gemessen und verglichen.

2 Datengrundlage

Als Datengrundlage dient ein Web Scraping¹ von Autowebsites, das aus über 32.000 Autos mit 115 Spezifikationen besteht. Die ursprünglichen Daten waren nur als CSV-Datei öffentlich verfügbar und mussten dementsprechend vorverarbeitet werden. In einem ersten Schritt wurde die CSV-Datei mithilfe von einem Skript in Python zu einer XML (ähnlichen)-Struktur konvertiert. Die erste Spalte wurde zu XML Elementen überführt und alle anderen Spalten bzw. Zeilen zu den dazugehörigen Werten gemacht. Leider enthielten die so generierten XML-Elemente Zeichen, die in den Namenskonventionen für XML-Elemente nicht zulässig sind (wie Leerzeichen oder bestimmte Sonderzeichen). Mithilfe von regulären Ausdrücken war es jedoch möglich, diese zu entfernen und ein wohlgeformtes XML-Dokument zu erhalten.

3 Modellierungsentscheidungen

In diesem Projekt wird ein besonderes Augenmerk auf den Vergleich zwischen Element und Attribut in XML gelegt. Es ist möglich, dieselbe Menge an Information innerhalb von Elementen oder innerhalb von Attributen darzustellen. Elemente und Attribute sind aber nicht identisch und können unterschiedliche Funktionen erfüllen (Attribute können beispielsweise keine Strukturen beschreiben). Trotzdem gibt es keine Regeln für die Nutzung von Elementen und Attributen, jedoch wird allgemein davon abgeraten², nur Attribute zum Modellieren in XML zu benutzen. Ein Grund dafür ist vor allem die erschwerte Lesbarkeit, was nachfolgend gezeigt wird. Nachdem die ursprüngliche CSV-Datei zu einer XML-Datei konvertiert wurde und einige Elemente angepasst wurden, hat die erste Modellierung der Autodatenbank folgende Struktur:

¹<https://github.com/nicolas-gervais/predicting-car-price-from-scraped-data>

²“Try to use elements to describe data” (https://www.w3schools.com/xml/xml_dtd_el_vs_attr.asp)

```

<?xml version="1.0" encoding="UTF-8"?>
<car_database
xmlns="https://www.w3schools.com/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://www.w3schools.com/ schema.xsd">
  <car>
    <Model>2019 Acura RDX Specs FWD w/Technology Pkg</Model>
    <MSRP_in_USD>40.600</MSRP_in_USD>
    <Gas_Mileage>22 mpg City/28 mpg Hwy</Gas_Mileage>
    <Engine>Turbo Premium Unleaded I-4. 2.0 L</Engine>
    [...]
    <Style_Name>FWD w/Technology Pkg</Style_Name>
    <Drivetrain>Front Wheel Drive</Drivetrain>
    <Passenger_Capacity>5</Passenger_Capacity>
    <Passenger_Doors>4</Passenger_Doors>
  </car>
</car_database>

```

Wie man bereits schnell erkennen kann, ist jede Information über ein bestimmtes Auto in Elementen verpackt. Zudem wird die XML-Version und die Zeichenkodierung in der ersten Zeile deklariert und in den nächsten Zeilen die Namensräume definiert und auf das dazugehörige Schema verwiesen. Ausgehend von dieser Struktur wurden diese Informationen innerhalb der Elemente mithilfe der *eXtensible Stylesheet Language Transformation* (XSLT) zu Werten in Attributen transformiert.

```

<?xml version="1.0" encoding="UTF-8"?>
<car_database
xmlns="https://www.w3schools.com/"
xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
xsi:schemaLocation="https://www.w3schools.com/ schema_attr.xsd">
  <car Model="2018 BMW i3 Specs 94 Ah" MSRP_in_USD="40.600" [...] </car>
  [...]
  <car Model="2019 Porsche 718 Specs Roadster" MSRP_in_USD="59.000" [...] </car>
</car_database>

```

Wie schon erwähnt, befinden sich die Werte der Spezifikationen in den Attributen des <car>-Elementes. Dies führt dazu, dass die Zeilen sehr lang werden³ und es schwierig wird, die XML-Datei leicht zu lesen. Die zweite Modellierungsentscheidung hat also schon

³Um die Übersichtlichkeit zu bewahren, wurde in den Ausschnitten die XML Struktur mit "[...]" verkleinert

mal einen Nachteil in der Lesbarkeit gegenüber der ersten Modellierung nur mit Elementen. In diesem Projekt soll jedoch die Laufzeit verglichen werden, ganz abgesehen von der Lesbarkeit des XML-Dokumentes.

Zudem sind die vorliegenden XML-Strukturen ohne Namensräume definiert. So kann es zwar zu Problemen in der Modularität kommen, jedoch sind Namensräume in meinem Anwendungsbeispiel nicht zwingend notwendig. Für weitere Vorgehensweisen sollte man jedoch Namensräume und die dazugehörigen Präfixe definieren.

Die valide Struktur der zwei XML-Dokumente muss separat jeweils in einer *XML Schema Definition* (XSD) definiert werden. Es wurde sich bewusst gegen eine alternative *Document Type Definition* (DTD) entschieden, da ein XML Schema es möglich macht, (eigene) Datentypen in Element- und Attributdeklarationen zu definieren. Dies ist insofern wichtig, da sich viele Spezifikationen in dem Datentyp unterscheiden. Zum Beispiel haben einige Spezifikationen den Datentyp *xs:string* (wie Bezeichnungen des Modells, des Motors etc.), aber es gibt auch Spezifikationen, die nur positive Dezimalzahlen annehmen können (wie das Leergewicht) oder nur positive, ganze Zahlen (wie die Passagierkapazität oder die Anzahl der Türen). Außerdem wird jede Spezifikation als optional angesehen, außer die Modellbezeichnung des Autos. Grund für diese Entscheidung war, dass es so möglich ist, die Datenbank mit Autos zu erweitern, auch wenn bestimmte Spezifikationen eventuell gar nicht verfügbar sind. Das einzige, was bei einer Erweiterung obligatorisch sein soll, ist die Modellbezeichnung.

Für die zwei XML-Dokumente liegen zwei Schemata vor, *schema.xsd* für die Modellierung nur mit Elementen und *schema_attr.xsd* für die Modellierung mit Attributen. Bis auf die Element bzw. Attribut eigene Syntax sind diese Schemata inhaltlich identisch.

4 XPath Abfragen

Da in diesem Projekt hauptsächlich die Laufzeit von XPath Ausdrücken untersucht werden soll, muss die Verarbeitungszeit natürlich auch gemessen werden. Dafür bietet sich das XML-Datenbankmanagementsystem *BaseX*⁴ an, welches zur Visualisierung und zur Abfrage von großen XML-Dokumenten entwickelt wurde. In diesem Programm ist es möglich durch den XQuery 3.1 Prozessor XPath Anfragen zu tätigen und sich die Verarbeitungszeit ausgeben zu lassen (siehe Abbildung 1).

Die Gesamtlaufzeit setzt sich also aus den Bereichen *Parsing*, *Compiling*, *Evaluating* und *Printing* zusammen. Inwiefern sich die XPath Ausdrücke in den einzelnen Abschnitten unterscheiden, wird im Folgenden dargestellt. Die Messung wurde für jede Suchanfrage zehn Mal durchgeführt⁵ und am Ende wurde der Durchschnitt berechnet, um mögliche Ausreißer bzw. Ungenauigkeiten zu vermeiden.

⁴<https://basex.org/>

⁵Hardware: AMD Ryzen 5 2600 6x 3.40GHz und 16 GB DDR4 RAM

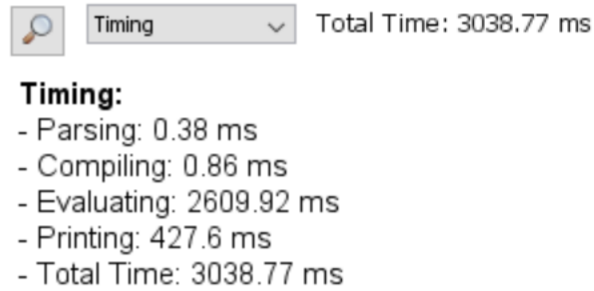


Abbildung 1: Angabe der Laufzeit in BaseX

4.1 XPath Abfrage 1

In der ersten Abfrage sollen Bezeichnungen der Motoren der ersten zehn Autos in der Datenbank ausgegeben werden. Der absolute Pfad für die beiden XPath Ausdrücke lautet `/car_database/car`, gefolgt von dem Prädikat `[position()<=10]`, welches nur die ersten zehn Resultate ausgibt. Abgeschlossen wird der Ausdruck mit `/Engine` (für die Struktur mit Elementen) oder `@Engine` (für die Struktur mit Attributen). In Tabelle 1 wird die Laufzeit der beiden Ausdrücke dargestellt.

XPath Ausdruck	Parsing	Compiling	Evaluating	Printing	Gesamtlaufzeit	Differenz
<code>/car_database/car[position()<=10]/Engine</code>	0,171 ms	0,245 ms	0,626 ms	0,123 ms	1,185 ms	-0,172 ms
<code>/car_database/car[position()<=10]/@Engine</code>	0,203 ms	0,248 ms	0,499 ms	0,067 ms	1,013 ms	+0,172 ms

Tabelle 1

Die Gesamtlaufzeit des XPath-Ausdrucks des XML Dokumentes mit Attributen ist 0,172 ms schneller als die Verarbeitungszeit bei der Modellierung nur mit Elementen. Die Differenz ist jedoch nur gering und lässt noch keine eindeutigen Aussagen zu.

4.2 XPath Abfrage 2

Es sollen die Autos in der Datenbank gefunden werden, die einen Listenpreis von über \$90.000 haben. Der XPath Ausdruck setzt sich in beiden Modellierungsfällen aus dem absoluten Pfad `/car_database/car` und dem Prädikat `[MSRP_in_USD>90.000]` bzw. `[@MSRP_in_USD>90.000]` zusammen.

XPath Ausdruck	Parsing	Compiling	Evaluating	Printing	Gesamtlaufzeit	Differenz
<code>/car_database/car[MSRP_in_USD>90,000]</code>	0,185 ms	0,339 ms	1056,043 ms	211,555 ms	1268,121 ms	-564,111 ms
<code>/car_database/car[@MSRP_in_USD>90,000]</code>	0,173 ms	0,319 ms	591,765 ms	111,751 ms	704,01 ms	+564,111 ms

Tabelle 2

Die Tabelle 2 zeigt sehr deutlich, dass die Struktur mit Attributen fast doppelt so schnell wie die Struktur mit Elementen verarbeitet wird. Was außerdem auffällt, ist, dass die Differenz zwischen den Gesamtlaufzeiten vor allem beim Evaluieren gebildet wird. So wie es aussieht, können Attribute schneller evaluiert werden als Elemente.

4.3 XPath Abfrage 3

In dieser Abfrage sollen Prädikate mit logischen Operatoren verbunden werden, sodass der resultierende Ausdruck komplexer wird. Ziel ist es, Autos zu finden, die einerseits Platz für mehr als sieben Leute haben und andererseits nicht über \$60.000 kosten. Der absolute Pfad bleibt logischerweise gleich, gefolgt von den zwei Bedingungen, die mit dem logischen Operator *and* verbunden werden: *[MSRP_in_USD<60.000 and Passenger_Capacity>7]* bzw. *[@MSRP_in_USD<60.000 and @Passenger_Capacity>7]*.

XPath Ausdruck	Parsing	Compiling	Evaluating	Printing	Gesamtlaufzeit	Differenz
/car_database/car[MSRP_in_USD>90,000]	0,299 ms	0,523 ms	1069,781 ms	223,746 ms	1294,355 ms	-134,373 ms
/car_database/car[@MSRP_in_USD>90,000]	0,206 ms	0,465 ms	1007,619 ms	151,696 ms	1159,982 ms	+134,373 ms

Tabelle 3

Tabelle 3 bestätigt zwar wieder die Tendenz, dass Attribute schneller verarbeitet werden, jedoch in diesem Fall nicht so eindeutig wie in Tabelle 2. Der größte Unterschied lässt sich beim Printing erkennen, sonst ist es nicht so eindeutig (aber in jedem Fall schneller).

4.4 XPath Abfrage 4

Die vierte Suche zielt darauf ab, Autos von der Marke Porsche zu finden, die das Baujahr 2018 und Platz für mehr als zwei Personen haben. Dieser komplexere Ausdruck besteht aus zwei logischen Operatoren und den Funktionen *starts-with()* und *contains()*. Mit Hilfe dieser Funktionen kann man gezielt die vorliegende XML Struktur ausnutzen. Im Falle von *starts-with()* wird so das Baujahr herausgefiltert, da der Inhalt des <Model>-Elements (bzw. des Model-Attributes) nach folgendem Schema aufgebaut ist: "[Baujahr] [Marke] [Modell] [Mögliche Spezifikation]". Jeder Wert startet also mit dem Baujahr und beinhaltet die Marke, die mit der *contains()*-Funktion gesucht wird.

XPath Ausdruck	Parsing	Compiling	Evaluating	Printing	Gesamtlaufzeit	Differenz
/car_database/car[starts-with(Model,'2018') and contains(Model,'Porsche') and Passenger_Capacity>2]/Model	0,318 ms	0,505 ms	1170,491 ms	0,893 ms	1172,229 ms	-528,74 ms
/car_database/car[starts-with(@Model,'2018') and contains(@Model,'Porsche') and @Passenger_Capacity>2]/Model	0,262 ms	0,508 ms	642,441 ms	0,422 ms	643,633 ms	+528,74 ms

Tabelle 4

Das Resultat ist sehr ähnlich zu der Verarbeitungszeit aus Abfrage 2. Attribute werden in dieser Suche 528,74 ms schneller verarbeitet, was wieder fast doppelt so schnell ist wie die Struktur nur mit Elementen. Hier sticht auch die Zeit des Evaluierens besonders ins Auge, aber auch die *Printing* Zeit.

4.5 XPath Abfrage 5

In der letzten Abfrage soll ein Durchschnittswert berechnet werden. Man möchte den durchschnittlichen Preis eines BMWs wissen, der entweder Allrad- oder Hinterradantrieb

hat. XPath Ausdrücke unterstützen mathematische Operatoren und Funktionen, wie die Funktion *avg()*. Der passende XPath Ausdruck wird mit den logischen Operatoren *and* und *or* gebildet, wobei der *or* Operator erst geklammert werden muss.

XPath Ausdruck	Parsing	Compiling	Evaluating	Printing	Gesamtlaufzeit	Differenz
<i>avg(/car_database/car[contains(Model, 'BMW') and (contains (Drivetrain, 'Rear Wheel Drive') or contains (Drivetrain, 'All Wheel Drive'))]/MSRP_in_USD)</i>	0,374 ms	0,466 ms	1190,382 ms	0,078 ms	1191,3 ms	-533,153 ms
<i>avg(/car_database/car[contains(@Model, 'BMW') and (contains (@Drivetrain, 'Rear Wheel Drive') or contains (@Drivetrain, 'All Wheel Drive'))]/@MSRP_in_USD)</i>	0,331 ms	0,477 ms	657,257 ms	0,081 ms	658,147 ms	+533,153 ms

Tabelle 5

Diese komplexe Suche bzw. Berechnung mit XPath bestätigt endgültig die Annahmen der zuvor abgefragten Suchen. Die Laufzeit der Abfrage über Attribute beträgt 658,147 ms, während die Laufzeit über Elemente 1191,3 ms dauert. Dieser Unterschied bildet sich vor allem beim Evaluieren. Tabelle 5 zeigt tatsächlich, dass die zwei Ausdrücke sich in der Laufzeit nur großartig beim Evaluieren differenzieren.

4.6 Sonstige XPath Abfragen

Die oben gezeigten XPath Ausdrücke haben eine Gemeinsamkeit, die bei der Bestimmung der Laufzeit eine wichtige Rolle spielen. Sie starten immer mit dem absoluten Pfad */car_database/car*. Zusätzlich gibt es aber noch die Möglichkeit, relative Pfade, die nicht mit dem Wurzelement beginnen, zu definieren. In diesem Beispiel wäre der relative Pfad *//car* (direkt gefolgt von einem Prädikat). Da dies zwei mögliche Strukturen innerhalb der XPath Ausdrücke darstellt und sich nicht direkt auf die Modellierungsentscheidung Element vs. Attribut bezieht, wurden diese Strukturen in dem Projekt weniger intensiv untersucht. Nichtsdestotrotz wurden diese zwei Alternativen getestet und die Ergebnisse in Tabelle 6 festgehalten.

XPath Ausdruck	Parsing	Compiling	Evaluating	Printing	Gesamtlaufzeit	Differenz
<i>//car[contains(Model, 'Porsche')]/Model</i>	0,21 ms	0,51 ms	2075,46 ms	6,61 ms	2082,79 ms	-960,47 ms
<i>/car_database/car[contains(Model, 'Porsche')]/Model</i>	0,27 ms	0,23 ms	1115,02 ms	6,8 ms	1122,32 ms	+960,47 ms
<i>//car[contains(@Model, 'Porsche')]/@Model</i>	0,21 ms	0,47 ms	657,76 ms	2,84 ms	661,27 ms	+44,09 ms
<i>/car_database/car[contains(@Model, 'Porsche')]/@Model</i>	0,25 ms	0,26 ms	613,66 ms	3,0 ms	617,17 ms	-44,09 ms

Tabelle 6

Insbesondere der relative XPath Ausdruck der XML Struktur nur mit Elementen zeigt eine höhere Verarbeitungszeit. In den Bereichen *Compiling* und *Evaluating* ist dieser fast doppelt so langsam wie der absolute Ausdruck. Im Vergleich dazu zeigen die absoluten und relativen Ausdrücke für Attribute nahezu keinen Unterschied.

Was in XPath zusätzlich möglich ist, sind ausführliche Notationen mit vollständigen Achsenbezeichnungen. Der Ausdruck */car_database/car[MSRP_in_USD > 90.000]* aus Tabelle 2 lässt sich ausführlich in dieser Form notieren: */car_database/child::car[MSRP_in_USD > 90.000]*. Diese Achsenbezeichnungen funktionieren natürlich auch für Attribute. Außerdem kann man Attribute ausführlicher schreiben, wenn man anstelle von *@MSRP_in_USD* die al-

ternative Schreibweise `attribute::MSRP_in_USD` einfügt. Die Tests hierzu haben jedoch gezeigt, dass eine ausführliche Notation keinen Einfluss auf die Laufzeit hat.

5 Visualisierung der Ergebnisse

Da ich die Ergebnisse der XPath Ausdrücke in einer HTML Seite visualisieren wollte, transformiert die XSLT-Datei `to_html.xsl` die XML Datenbank `car_database.xml` zu einer HTML-Seite. Dabei wurden die Tabellen mit CSS gestaltet und durch eine Scrollbar übersichtlich gemacht. Über die Zellen in den Tabellen wurde durch die `<xsl:for-each select="[XPath Ausdruck]">` Schleife iteriert und mit dem passenden Wert `<xsl:value-of select="[XPath Ausdruck]"/>` aufgefüllt.

6 Modellierungen im Überblick

Diese Grafik stellt einen kurzen Überblick über die verschiedenen XML-Dateien und ihre Beziehungen zueinander dar.

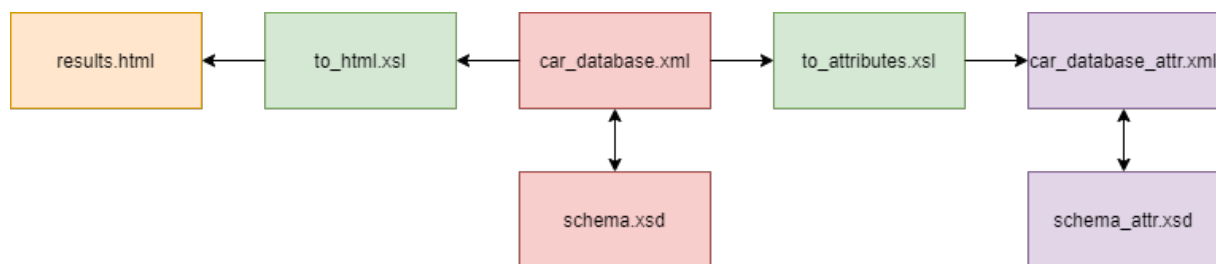


Abbildung 2: Übersicht über die im Projekt erstellten Dateien

7 Fazit

Die Testergebnisse haben eindeutig gezeigt, dass die Laufzeit von XPath Anfragen an eine XML Datenbank mit Werten, die nur in Attributen verpackt sind und nicht in Elementen, am geringsten ist und dies somit die effizientere Struktur dargestellt. In den meisten Fällen unterscheiden sich diese beiden Strukturen gerade in der Evaluierungszeit und das *Printing* der Resultate läuft in der Regel ebenfalls schneller ab. Wenn man sich das Ergebnis anguckt, könnte man die Aussage aufstellen, dass man immer anstelle von Elementen Attribute nutzen sollte (wenn man nur Werte abbilden möchte und keine verschachtelten Strukturen etc.). Diese Aussage würde ich persönlich aber nicht unterstreichen. Ein wichtiger Vorteil von XML ist die Möglichkeit, eine Struktur zu definieren, die sowohl maschinen- als auch menschenlesbar ist. Wie in meiner zweiten Modellierung schon gezeigt, sind Strukturen die quasi ausschließlich aus Attributen bestehen, nicht so leicht zu

lesen. Dieses Merkmal würde ich als wichtiger gewichten als den Vorteil einer schnelleren Verarbeitung. Gerade wenn mehrere Menschen an einer XML-Datenbank arbeiten und sie pflegen müssen, ist die Lesbarkeit extrem wichtig. In dem vorliegenden Beispiel der Autodatenbank hätte ich eine andere Struktur gewählt, die intuitiver gestaltet ist und eine sinnvolle und logische Nutzung von Attributen und Elementen vorweist. Beispielsweise lassen sich Attribute sehr gut für die Annotation von Metadaten verwenden. Falls man jedoch eine XML-Struktur braucht, die so schnell wie möglich verarbeitet werden soll, sollte man so viele Attribute wie möglich verwenden und dort, wo es möglich ist, absolute Pfade anstelle von relativen Pfaden verwenden.

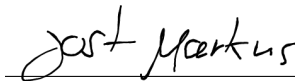
8 Ausblick

Die Autodatenbank lässt sich unter Berücksichtigung des XML Schemas sehr leicht erweitern. Gerade die Spezifikationen könnte man verbessern, indem man beispielsweise mehr Datentypen definiert und bestimmte Spezifikationen hinzufügt, die sich wiederum aus Anderen ergeben (zum Beispiel könnte man die Spezifikation "SAE_Net_Horsepower_RPM" in kW umrechnen und dabei eine neue Spezifikation bilden, die noch nicht vorhanden ist). Dabei wäre es jedoch sinnvoll, Namensräume zu definieren, um die Modularität zu sichern. Abgesehen davon könnte man die XML-Datenbank mithilfe von XSLT detaillierter visualisieren und mithilfe von Javascript interaktiver gestalten. Ursprünglich war auch geplant, mit Javascript eine Eingabe zu ermöglichen, die entweder einen XPath Ausdruck direkt im Browser entgegennimmt oder nach bestimmte Spezifikationen suchen lässt und die Wünsche des Benutzers in einen XPath Ausdruck verwandelt und dann eine Anfrage an die Datenbank durchführt. Diese Idee ist für den vorgesehenen Umfang des Projektes jedoch ungeeignet und erfordert einige Kenntnisse in Javascript, welches nicht Teil der Veranstaltungen Informationsstrukturierung bzw. Informationsstrukturierung 2 war.

Eigenständigkeitserklärung

Hiermit erkläre ich, dass ich den vorliegenden Projektbericht selbständig verfasst und gelieferte Datensätze, Zeichnungen, Skizzen und graphische Darstellungen selbständig erstellt habe. Ich habe keine anderen Quellen als die angegebenen benutzt und habe die Stellen der Arbeit, die anderen Werken entnommen sind - einschl. verwendeter Tabellen und Abbildungen - in jedem einzelnen Fall unter Angabe der Quelle als Entlehnung kenntlich gemacht.

Bielefeld, den 31.08.2021

A handwritten signature in black ink, reading "Jost Markus", written over a horizontal line.

Jost Markus