Daten & XML

Christoph Rinne*

09. Januar 2022

Inhaltsverzeichnis

1 Grundlegende Information

Das gesamte Thema im Kontext dieser Handreichungen auch nur einführend vorzustellen ist nicht möglich. Es werden also diverse Aspekte nur angerissen oder erwähnt oder stehen nur als Schlagworte im Text. Ziel ist es, Ihnen das Thema "Daten & XML" im Kontext der Archäologie verkürzt vorzustellen und mit wenigen Beispielen Nutzungsszenarien aufzuzeigen.

1.1 Entwicklung und Forschungsgeschichte

Daten sammeln und nachnutzen, zur Recherche oder der Analyse, ist in den archäologsichen Fächern ein alltäglicher Vorgang. Meist sind es einzelne Tabellen, gelegentlich auch Tabellen mit Referenzen untereinander oder auch etwas komplexere relationale Datenbanken die hier zum Einsatz kommen. Daneben stehen aber auch andere Techniken zur Strukturierung von Informationen zur Verfügung, eine sehr grundlegende ist die Auszeichnungssprache XML.

Von dem eher technischen Aspekt der Sprache, z.B. XML oder SQL, ist das Konzept zur Strukturierung der Informationen zu trennen. Bedingt durch die jeweilige Enticklungsgeschichte stehen die vorgenannten Sprachen XML und SQL für sehr unterschiedliche Konzepte von Datenmodellen. SQL steht an den Anfängen der Datenbankentwicklung und für realtionale Datenbankmodelle. XML ist die jüngere und mit dem Internet eng verbundene Entwicklung, die zu semistrukturierten Daten in einer hierarchischen Baumstruktur führt.

Zu Beginn der 2000er wurde von Schloen ArchaeoML, ein auf XML basierendes und auf Objekte (*item*) zentriertes Datenmodell präsentiert (Schloen, 2001). Zeitgleich wird CIDOC-CRM entwickelt, welches ebenfalls auf XML und hieraus mit RDF als auch OWL entwickelten Relationen basierend, eine komplexe Ontologie für Informationen der Kutlurwissenschaften dartellt (Doerr u. a., 2003). Beide sind in ihrer Entstehung eng mit der Entwicklung des Internet verbunden und basieren im Kern auf XML, sind darüber hinaus aber sehr unterschiedlich (Kansa, 2005). ArchaeoML ist in OCHRE aufgegangen und

*Christian-Albrechts Universität zu Kiel, crinne@ufg.uni-kiel.de

bildet auch eine Grundlage von Open Context Hertiage Bytes, z.B. bei DINAA. Die weitere Entwicklung von Open Context erfolgt auf GitHub wobei GeoJSON-LD und damit der Raumbezug und die Referenzen unterienander zunehmend an Bedeutung gewinnen Kansa 2014. Auch CIDOC-CRM wird weiter entwickelt (Oldman, 2014) und vielfach als Grundlage für die Erfassung von Datenbeständen genutzt. Während CIDOC-CRM anfänglich stark an den Relationen orientiert war tritt nun auch hier das Objekt oder *item*, als Ding (*thing*), Objekt (*objekt*) oder (*feature*) verstärkt in das Zentrum.

Ausgehend von XML steht also das Element zunehmend im Zentrum der Strukturierung weniger die Relation von definierten Klassen wie es bei SQL eher der Fall ist. Weder XML noch SQL sind entgegen dieser sehr unterschiedlichen genuinen Konzeption aber auf die jeweiligen Modelle eingeschränkt.

1.2 XML Nutzen

Zur Nutzung von XML-Datenbanken wurde XQuery entwickelt. Grundlegendes Datenmodell ist eine Liste, mit keinem oder einem Element oder auch mehreren Elementen. Hierbei kann ein Element auch wiederum eine Liste sein. Diese als Sequenz bezeichnete Liste von Elementen entspricht damit der Struktur eines XML-Dokumentes. Bei XQuery spielen FLWOR-Ausdrücke ([flou-er], Blume) eine zentrale Rolle. FLWOR steht für *for, let, where, order by, return* und ist damit erkennbar an SQL orientiert (*SELECT, FROM, WHERE, ORDER BY*).

Eine Programm zur Nutzung von XML-Datenbanken und XQuery ist BaseX (s.u.).

1.3 XML in R / RStudio

Da ich diesen Text in R-Markdown mit RStudio schreibe verwende ich sogenannte *code chunks*, um die Befehle darzustellen. Mit den Paketen xml oder xml2 können XML-strukturierte Daten in R genutzt werden, dabei handelt es sich allerdings um sogenannte *wrapper* mit einer typischen R-Syntax und nicht um die Implementierung einer XML-Sprache wie z.B. XQuery. Beide Pakete haben jeweils eigene Dokumentationen (xml, xml2). Die Pakete bieten einige unterschiedliche Funktionen, z.B. xmlToDataFrame() bei xml und unterscheiden sich weitgehend in den Fundktionsnamen: “xml_<Beschreibung>” bei xml2.

XQuery wird in R nicht umgesetzt, die nachfolgenden *code chunks* mit XQuery sind demnach funktionsloses Layout ohne hervorgehobene Syntax. Für die Nutzung der Pakete xml und xml2 sind einführende Beispiele in der oben genannten Dokumentation vorhanden.

2 Software

XML-Daten und deren Nutzung sind eng mit dem Internet verbunden (s.o.). Die zugehörige Software setzt deshalb meist einen Server (z.B. *localhost*) und PHP voraus.

2.1 BaseX

BaseX ist plattformunabhängig, wird unter Open-Source-Lizenz (BSD) angeboten und für Windows gibt es ein eigenes Installationsprogramm. BaseX wird auf GitHub weiter entwickelt. Die Website des Projektes bietet eine umfassende Dokumentation, darunter auch eine knappe Einführung zu der nachfolgend genutzten GUI, die für mich der hervorzuhebende Aspekt dieser Software ist. Für den ersten Kontakt mit XML, XPath und XQuery empfehle ich auch den Text Basex for dummies von Paul Swennenhuis.

> Die GUI von BaseX erlaubt die Recherche in den Daten mit XPath und XQuery, das Editieren von Daten ist hier nicht möglich.

3 XML Grundlagen

XML besteht aus Elementen (s.o. Sequenz) die auch als Knoten eines Baumes bezeichnet und dargestellt werden. Jedes Element kann Attribute besitzen und einen Wert oder weitere Elemente beinhalten. Ein Beispiel ist:

```
<text autor="Sepp Herberger">
  <satz>
    <wort typ="artikel">Der</wort>
```

```
    <wort typ="substantiv">Ball</wort>
    <wort typ="verb">ist</wort>
    <wort typ="adjektiv">rund</wort>
    .
  </satz>
</text>
```

Das Wurzel-Element des Beispieles ist <text>...</text> mit den weiteren Elementen <satz> und <wort> die neben den *start tags* auch die zugehörigen *end tags* besitzen. Die Elemente <wort> beinhaltet konkrete Werte, ebenso <satz> mit dem Punkt. Das Element <text> besitzt das Attribut "autor" und das Element <wort> das Attribut "typ". Dabei ist das Attribut "typ" für <wort> gut gewählt, da es eine eindeutig Zusatzinformation ist, während wir für einen Text auch mehrere Autoren erwarten können und deshalb besser mit den Elementen <autoren> und <autor> arbeiten sollten.

Wenn Sie wohlgeformte komplexere XML-Dateien in einem Editor öffnen, z.B. KML-Dateien, sehen Sie als erstes eine Deklaration und auch Verweise auf anzuwendende Regelwerke. Dies dient der maschinellen Prüfung der Struktur und Daten.

```
<?xml version="1.0" encoding="UTF-8"?>
<kml xmlns="http://www.opengis.net/kml/2.2" xmlns:gx="http://www.google.com/kml/ext/2.2">
```

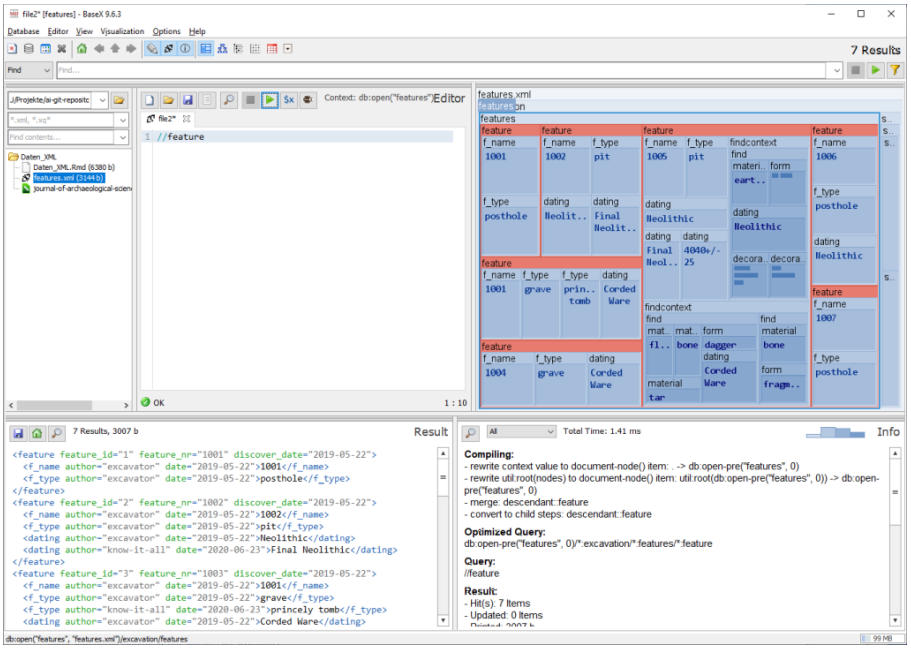

Abbildung 1: Die GUI von BaseX mit der geöffneten XML-Datenbank features

3.1 XML Beispieldaten

Starten Sie für die Arbeit mit den Beispieldaten das Programm **BaseX**. Mit "Database > New ..." öffnen Sie einen Ordner mit XML-Dateien oder eine einzelne XML-Datei und weisen dieser Datenbank einen Namen zu, z.B. "features" für die Beispieldaten features.xml. Mit "Databases > Open & Manage ..." können einmal angelegte Datenbanken erneut geladen werden. Ist die Datenbank "features" in BaseX geladen sollten Sie annähernd das vorangehende Bild sehen.

Unterhalb der üblichen Menü- und Icon-Leiste sehen Sie folgende Fenster: Projekt-Ordner, Editor für Syntax, *Map* der Datenbank, *Result* mit aktuell dem gesamten Datenbestand und *Info*. Diese Fenster können über das Menü unter *View* und *Visualization* auch an- und ausgeblendet werden.

Sehen Sie sich die Strutur der Daten sowohl im Fenster *Result* als auch *Map*-Fenster an. Innerhalb von "excavation" gibt es "features" und "samples". Ein feature kann zudem mit "findcontext" und "find" Fundzettelangaben und Funde beinhalten. Daneben stehen "samples" wobei eine Probe aus mehreren Funden zusammengesetzt sein kann und auf diese anhand einer "id" verweist. Zahlreiche Elemente haben Attribute zu dem Autor und dem Zeitpunkt der Wertezuweisung.

Diese Beispieldaten haben keine zureichende Strukturierung für die Daten einer realen Ausgrabung mithin auch keine Deklaration.

3.2 XPath

Mit XPath können die Knoten des Baumes adressiert werden, Rückgabe sind die Elemente und Werte ab der entsprechenden Ordnung. Tragen Sie bitte nacheinander und einzeln folgende XPath-Anweisungen in den Editor ein und führen diese aus:

```
//feature
//feature/dating
//dating
data(//dating)
```

Die Anweisungen liefern nacheinander:

- Die untergeordneten Knoten und Werte aller Befunde. Die "//" stehen für eine beliebige Position (Ordnung) im Baum.
- Liefert die Datierungen der Befunde.
- Liefert alle Datierungen im Datenbestand, hier der Befunde und Funde.
- Liefert ausschließlich die Werte aller Datierungen.

Daten können auch gefiltert werden.

```
//dating[@method]
//feature[@feature_nr > 1003]
//feature[@feature_nr > 1003]/f_type
//feature[f_type="grave"]
//feature[f_type = "grave" and f_name > 1002]
//feature[findcontext]
```

Die Anweisungen liefern nacheinander:

- Die Datierungen mit einem Attribut *method*.
- Alle Knoten und Werte der Befunde mit dem Attribut-Wert > 1003. Beachten Sie bitte, dass die Zahl im Datenbestand in " " steht und doch als Zahl evaluiert wird.
- Die Liste der Knoten *f_type* zum vorangehenden Filter.
- Alle Befunde vom Typ *grave*.
- Filter können mit *and*, *or* und () kombiniert werden.
- Alle Befunde die Fundkontexte und demnach wohl Funde (*//feature[findcontext/find]*) erbracht haben.

Auch unscharfe Suchen sind möglich, nachfolgend Beispiele für die Funktionen *matches()*, *contains()*.

```
//feature[contains(f_type,"i")]
//feature[matches(f_type,"^P","i")]
```

Die Anweisungen liefern nacheinander:

- Alle Befunde mit einem "i" im Typ. Die Funktion *contains()* hat zwei Parameter, das zu vergleichende Element und den gesuchten Ausdruck.
- Alle Befunde deren Typ mit einem "P" oder einem "p" beginnt. Die Funktion *matches()* hat drei Parameter: Das zu vergleichende Element, den gesuchten Ausdruck und optional die Anweisungen "i" oder "s" für *case insensitive / sensitive*. Das Zirkumflex entstammt in seiner Bedeutung als Zeilenanfang den Regulären Ausdrücken.

Sequenzen von Elementen müssen zerlegt werden.

```
//feature[dating[contains(text(), "Neol")]]
//feature[dating[matches(text(), "^Final", "s")]]
```

Die Beispieldaten haben mit *dating* auch mehrere Aussagen zur Datierung. Bei dem Befund 1005 ist es nach der Erstdatierung von der Ausgrabung einerseits eine feinere typologische Ansprache nach der Fundanalyse durch einen Spezialisten und eine ergänzende Radiokarbondatierung. Der einfache Filter für

die Befunde auf das Kind-Element *dating* liefert bei diesen Befunden eine Sequenz (Liste) von Werten (*Neolithic, Final Neolithic,...*) an der die Funktionen scheitern.

Die Funktion *text()* liefert nur die Werte des zu vergleichenden Elementes. Damit ist es auch möglich, die einzelnen Werte der Elemente *dating* zu vergleichen.

3.3 XQuery

Schließen Sie die Datenbank mit "Database > Close". Wir müssen ab jetzt die Quelle angeben, das geht unter anderem mit der Funktion *collection()*.

```
let $results :=
collection("features")//feature[dating[matches(text(), "^Final", "s")]]
return <results>{$results}</results>
```

In der vorangehende Anweisung wird die Rückgabe des vorab geschriebene Filters der Variablen *$results* zugewiesen und am Ende mit *return* zusammen mit dem neuen Tag <results> ... </results> zurückgegeben.

Wir Verändern die Anweisung im Stiel von FLOWER mit den Abschnitten *for*, *in*, der auswertenden Zuweisung *let*, *where* als Filter und *order by* für die Sortierung.

```
for $feature
in collection("features")//feature
where $feature[dating[contains(text(), "Neol")]]
let $finds := count($feature/findcontext/find)
order by $finds descending, $feature/@feature_nr
return <Befund>
  <Befundnr>{$feature/data(f_name)}</Befundnr>
  <Fundeanzahl>{$finds}</Fundeanzahl>
  </Befund>
```

Sie erkennen wohin die Reise geht: Die Rückgabe wird ein durch Tags strukturierter Text, der zur Darstellung in einem Browser genutzt werden kann. Im Detail muss diese Anweisung nach den vorangehenden Erläuterungen nicht wirklich erklärt werden. Die Funktion *count()* ist selbsterklärend und die Anweisung *order by* kann mit *descending* entgegen dem impliziten *ascending* auch umgekehrt werden.

Machen wir aus dem Ganzen mal eine Tabelle, wobei ich grundlegende Kenntnisse in HTML voraussetze:

```
<table>
<tr><th>Befund-Nr.</th><th>Anzahl Funde</th></tr>
{for $feature
in collection("features")//feature
where $feature[dating[contains(text(), "Neol")]]
let $finds := count($feature/findcontext/find)
order by $finds descending, $feature/@feature_nr
return <tr>
  <td>{$feature/data(f_name)}</td>
  <td>{$finds}</td>
  </tr>
}
</table>
```

Das Ergebnis sieht so aus:

```
<table>
  <tr>
    <th>Befund-Nr.</th>
    <th>Anzahl Funde</th>
  </tr>
  <tr>
    <td>1005</td>
    <td>3</td>
```

```
  </tr>
  <tr>
    <td>1002</td>
    <td>0</td>
  </tr>
  <tr>
    <td>1006</td>
    <td>0</td>
  </tr>
</table>
```

Es kann im Browser umgesetzt und korrekt angezeigt werden, etwa so:

Befund-Nr.	Anzahl Funde
1005	3
1002	0
1006	0

Zitierte Literatur

Doerr, M., Hunter, J., Lagoze, C., 2003. Towards a Core Ontology for Information Integration. Journal of Digital Information 4.

Kansa, E., 2005. A Community Approach to Data Integration: Authorship and Building Meaningful Links across Diverse Archaeological Data Sets. Geosphere 1, 97–109. https://doi.org/10.1130/GES00013.1

Oldman, D., 2014. The CIDOC Conceptual Reference Model (CIDOC-CRM): PRIMER.

Schloen, J.D., 2001. Archaeological Data Models and Web Publication Using XML. Computers and the Humanities 35, 123–152. https://doi.org/10.1023/A:1002471112790