Extracting Output Schemas from XSLT Stylesheets and Their Possible Applications

Ruben Mendes
INESC-ID, Portugal
ruben.mendes@ist.utl.pt

José Borbinha
INESC-ID, Portugal
jlb@ist.utl.pt

Hugo Manguinhas
INESC-ID, Portugal
hugo.manguinhas@ist.utl.pt

Abstract

XML is nowadays the dominant standard used for data representation and exchanging. XML documents can be transformed into different formats by using the transformation language XSLT. XSLT stylesheets can be designed to present and transform XML input data in one schema in XML output data according to other schema. In this paper we describe how to compute automatically an output XML data schema given an XSLT stylesheet that is intended to produce it. The main objective of this work is to develop a tool that contributes to a better understanding of the XML transformation process. The first results, focused on bibliographic data, suggest that we can determine the output schemas of most common XSLT stylesheets.

Keywords: XML; XSLT; XML Schema; schema transformation

1. Introduction

Data exchanging is the key operation when partners cooperate for business. Whenever two or more exchange partners use different data formats, than at least one of the partners will need to transform the data. The exchange of documents in e-commerce suffers from heterogeneity problems at several levels, from code choice to document structure or even semantic differences (Edger & Strametz , 2001). XML (eXtensible Markup Language) (W3C, 2008) solved many of those problems and is nowadays the dominant standard used for data exchanging and representation on the Web. Therefore, XSLT (eXtensible Stylesheet Language Transformations) (Kay, 2007) was created as a language to declare transformations of XML documents.

XSLT is used to grab, filter and associate data from XML documents. However, XSLT is considered a language difficult to learn (Bac & Bailey, 2003), so it is relevant if we can develop tools to assist in scenarios where those difficulties can be a limitation.

In this paper we describe how to compute the output XML Schema (W3C, 2001) of a given XSLT stylesheet. Our results show that in some circumstances we can successfully determine the output schema of a given XSLT stylesheet.

Determining successfully the output schema contributes to better understanding of an XML transformation process, which can be helpful in several application scenarios, such as:

- *Scenario 1:* Having an XSLT stylesheet, we face the need to update it when the output schema is modified.
- *Scenario 2:* Having a data file resulted from a XSLT transformation, determining the coverage of that data according to the intended output schema.
- *Scenario 3:* Having two different written XSLT stylesheets, measure how extent they produce the same output schema.

Our purpose is to address these three scenarios, and thus here we report our first findings towards that.

The following of this document is structured as follows: Section 2 provides an overview of the proposed solution. In Section 3 we present our experimental results. We end up with the conclusion and future work in Section 4.

OPEN ACCESS
This article is licensed under a **Creative Commons Attribution 4.0 International License**, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and cite the source.

https://doi.org/10.23106/dcmi.952136056

2. Proposed Solution

Ignoring so far the data resultant of the XSLT transformation process, we compute automatically the output schema based on only a given XSLT stylesheet. For those cases, we determine automatically the output schema of the given XSLT stylesheet by performing a static analysis on the XSLT nodes. The Figure 1 represents the architecture for that solution.

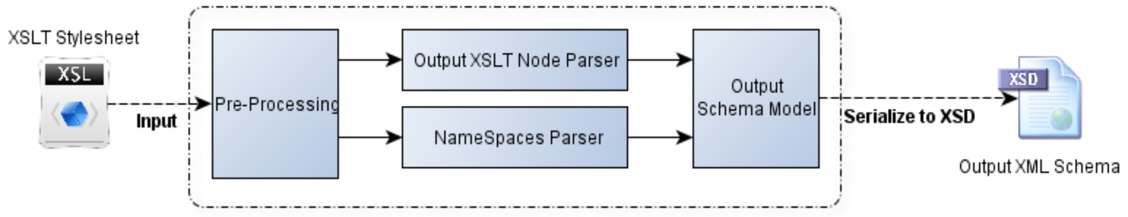


FIG. 1: Solution Architecture

For the determination of the output schema, first we load the XSLT stylesheet into memory and structure it in a way where each XSLT node is modeled as an object. Afterwards, all stylesheet namespaces are extracted and then all output XSLT nodes are identified. For the calculation of the output schema model we developed the algorithm getOuputModel, which, given an XSLT stylesheet as input, returns an internal model representation of the output schema. The main part of this algorithm is represented in Algorithm 1.

The output nodes to be identified are those nodes that:

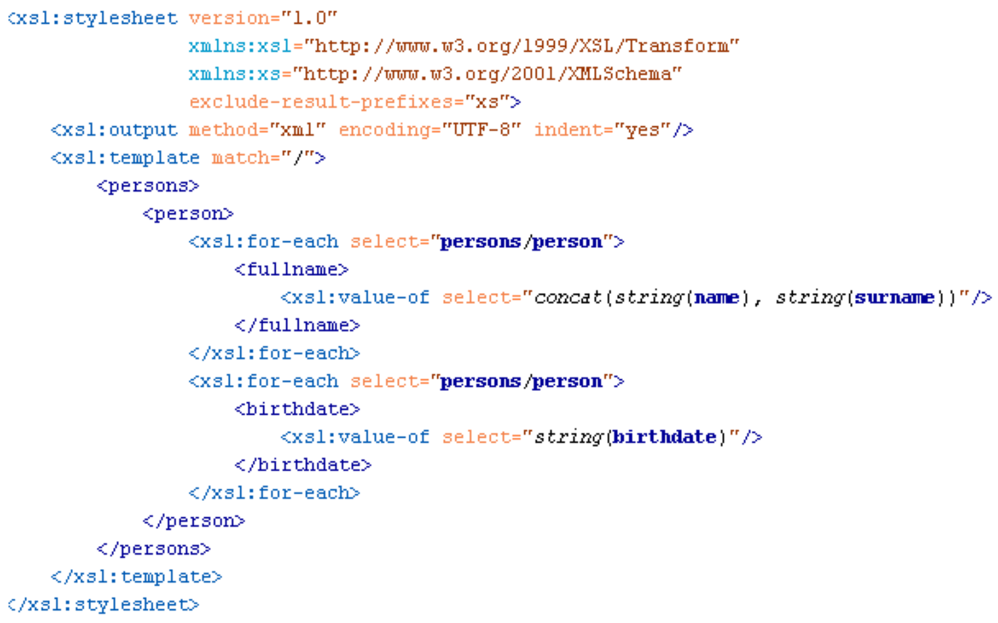
- generate an element <elementname>
- attach one attribute to the respective element node <xsl:attribute>

```
Input: XSLT Stylsheet
Output: XML Schema
outputModel output= new outputModel();
level=0;
foreach XSLTNode  do
    if: (XSLTNode = = (XSLTElement or XSLTAttribute)) then
        level=level + 1;
        getOutputModel();
        if (outputModel does not contain XSLTnode)    then
            output.add(XSLTNode,level);
        end
        level = level - 1;
    end
end
writeXMLSchema(output);
```

ALGORITHMI 1: The algorithm getOutputModel

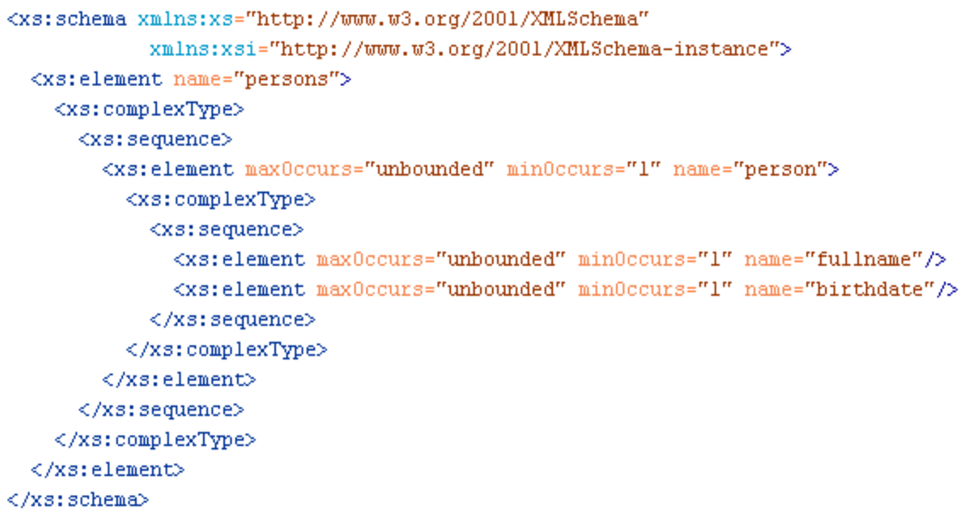
In Figure 2 we show and example of an input, which produced as output the XML Schema shown in the Figure 3. We conclude that this XSLT stylesheet basically receives an input XML file with elements firstName, surName and birthdate and returns as output an XMLfile with elements fullName and birthdate. The output XML Schema uses the output elements as detected in the XSLT stylesheet by our technique.

OPEN ACCESS
This article is licensed under a **Creative Commons Attribution 4.0 International License**, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and cite the source.



```
<xsl:stylesheet version="1.0"
                xmlns:xsl="http://www.w3.org/1999/XSL/Transform"
                xmlns:xs="http://www.w3.org/2001/XMLSchema"
                exclude-result-prefixes="xs">
    <xsl:output method="xml" encoding="UTF-8" indent="yes"/>
    <xsl:template match="/">
        <persons>
            <person>
                <xsl:for-each select="persons/person">
                    <fullname>
                        <xsl:value-of select="concat(string(name), string(surname))"/>
                    </fullname>
                </xsl:for-each>
                <xsl:for-each select="persons/person">
                    <birthdate>
                        <xsl:value-of select="string(birthdate)"/>
                    </birthdate>
                </xsl:for-each>
            </person>
        </persons>
    </xsl:template>
</xsl:stylesheet>
```

FIG. 2: XSLT stylesheet concat.xsl



```
<xs:schema xmlns:xs="http://www.w3.org/2001/XMLSchema"
           xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance">
  <xs:element name="persons">
    <xs:complexType>
      <xs:sequence>
        <xs:element maxOccurs="unbounded" minOccurs="1" name="person">
          <xs:complexType>
            <xs:sequence>
              <xs:element maxOccurs="unbounded" minOccurs="1" name="fullname"/>
              <xs:element maxOccurs="unbounded" minOccurs="1" name="birthdate"/>
            </xs:sequence>
          </xs:complexType>
        </xs:element>
      </xs:sequence>
    </xs:complexType>
  </xs:element>
</xs:schema>
```

FIG. 3: Output Schema concat.xsd

3. Experimental Results

Our algorithm was implemented using Java. The first tests were done with a collection of 13 XSLT stylesheets from Portuguese libraries, archives and museums that are data providers for Europeana1.

This XSLT collection contains stylesheets that transform several data schemas, such as MarcXchange (for data files in UNIMARC and MARC21), OAI-DC and other local defined schemas into the ESE (Europeana Semantic Elements) , as follows:

- The data in cases 6, 8, 12 and 13 is UNIMARC, coded in MarcXchange.
- The data in case 5 is in OAI-DC.
- The data in case 6 is MARC-21, coded in MarcXchange.
- The remaining data is coded in individually local defined schemas.

OPEN ACCESS
This article is licensed under a **Creative Commons Attribution 4.0 International License**, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and cite the source.

We realized that the XSLT stylesheets contained in this collection make use of a large number of coding options offered by the XSL language.

Our solution successfully extracted the output schemas of 13 out of the 13 XSLT stylesheets, as reported in Table 1.

TABLE. 1: Comparing to Original Europeana XML Schema (made of 24 elements)

Stylesheet	1	2	3	4	5	6	7	8	9	10	11	12	13
ESE Coverage	14	11	11	5	13	20	12	19	16	11	10	21	20
Extra Elements	2	1	1	0	1	4	1	5	0	2	2	4	4

We realized that, for the resultant output schemas having missing ESE elements, they don't have that information available on the input XML data because that information already is not to be expected in the input data (it is not relevant in the organization the produced that data). Regarding the extra elements found, which are not part of the ESE schema, mainly related with the UNIMARC cases, we can assume that they contain vital information that the programmer of the XSLT decide to express in the output, but it isn't representable by the ESE schema.

4. Related Work

To the best of our knowledge there is only one contribution for the calculation of output schemas from XSLT stylesheets described in S. Groppe and J. Groppe (2008). This approach is very similar to our method with the main difference being the pre-processing and memory representation where they load the xslt nodes into an abstract syntax tree similar to the DOM structure. We find our approach less complex and easier to understand.

There is also the work done in Groppe, Böttcher, Birkenheuer and Höing (2006), which deals with the search on the output nodes of XSLT stylesheets as part of a static analysis for the optimization of the execution time of XSLT stylesheets.

5. Conclusions and Future Work

Our solution will be integrated in REPOX framework (Reis et al., 2009). REPOX is a framework to preserve and manage data sets, with are expected to be mainly bibliographic sets. REPOX can play the role of a broker or other specific service in a Service Oriented Architecture (it has built-in servers and clients for HTTP, OAI-PMH, FTP, etc.), to manage, transparently, data sets of information entities in digital libraries, independently of their schemas or formats. The main default functions of this framework are data set submission, data set storage, data set transformations and data set retrieval. REPOX is used in the infrastructures of many data providers for Europeana and TEL - The European Library .as also in the internal infrastructures of these entities.

We have proposed an approach for the determination of the output schema of any XSLT stylesheet, which is understood as the schema to which all possible results of the XSLT stylesheet conform to. We have described the architecture of our solution that is capable of computing any output schema for the most typical XSLT stylesheets.

This is work in progress, which we expect to complement with techniques to support processes for XSLT reverse engineering. The ultimate purpose is to make it possible, in REPOX, to create visual mapping representations more suitable for human analysis given one XSLT stylesheet. Such a tool will be extremely useful for the professional in Europeana, TEL, and their data providers (but also to students, web developers or programmers that are in process of learning XSLT). We expect that our solution will contribute to a better understanding of the XML transformation processes.

OPEN ACCESS
This article is licensed under a **Creative Commons Attribution 4.0 International License**, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and cite the source.

https://doi.org/10.23106/dcmi.952136056

References

Bac, E. and J. Bailey. (2003). CodeX: an approach for debugging XSLT transformations," In Proceedings of the 7th International Conference on Properties and Applications of Dielectric Materials, pp. 309-312.

Eder, J. and W. Strametz. (2001). Composition of xml-transformations." in EC-Web, ser. Lecture Notes in Computer Science, K. Bauknecht, S. K. Madria, and G. Pernul, Eds., vol. 2115. Springer, pp. 71-80

Groppe, S. and J. Groppe. (2008). "Output schemas of xslt stylesheets and their applications", Information Sciences, vol. 178, no. 21, pp. 3989-4018

Groppe, S., S. Böttcher, G. Birkenheuer, A. Höing. (2006) "Reformulating XPath queries and XSLT queries on XSLT views", Data and Knowledge Engineering Journal, 57 (1), pp. 64–110

Kay, M. (2007). XSL Transformations (XSLT), Ver.2.0

Reis, D., N. Freire, H. Manguinhas, and G. Pedrosa. (2009). REPOX - a framework for metadata interchange. In Research and Advanced Technology for Digital Libraries, ser. Lecture Notes in Computer Science, M. Agosti, J. Borbinha, S. Kapidakis, C. Papatheodorou, and G. Tsakonas, Eds. Springer Berlin / Heidelberg. vol. 5714, pp. 479-480

World Wide Web Consortium (W3C). (2001)..Fallside, D. C. (2001). XML Schema Part 0: Primer Second Edition, 2001, W3C Recommendation. D. C. Fallside and Priscilla Walmsley (eds). Available at http://www.w3.org/TR/xmlschema-0/

World Wide Web Consortium (W3C). (2008). Extensible markup language XML 1.0 5th edition, W3C Recommendation. C. M. E. M. T. Bray, J. Paoli, C. M. Sperberg-McQueen, E. Maler and F. Yergeau (eds). Available at http://www.w3.org/TR/xml/

OPEN ACCESS

This article is licensed under a **Creative Commons Attribution 4.0 International License**, which permits use, sharing, adaptation, distribution and reproduction in any medium or format, as long as you give appropriate credit to the original author(s) and cite the source.

https://doi.org/10.23106/dcmi.952136056