Streaming Transformation of XML to RDF using XPath-based Mappings

Jyun-Yao Huang
National Chung Hsing University, Taiwan
allen501pc@gmail.com

Christoph Lange
University of Bonn & Fraunhofer IAIS, Germany
langec@cs.uni-bonn.de

Sören Auer
University of Bonn & Fraunhofer IAIS, Germany
auer@cs.uni-bonn.de

ABSTRACT

The Extensible Markup Language (XML) has become a widely adopted data interchange format. With the rise of Linked Data published using the Resource Description Framework (RDF), a number of tools for transforming XML to RDF have been developed. Specifying XML→RDF mappings for these tools often requires skills in programming languages such as XSLT or XQuery. Moreover, these tools are rarely able to deal with large XML inputs. We introduce an XML to RDF transformation approach, which is based on mappings comprising RDF triple templates that employ simple XPath expressions. Thanks to the restricted XPath expressions, which can be evaluated against a stream of XML data, our implementation can handle extremely large input XML files. To process the XML input efficiently, we employ XML filtering techniques and a strategy for selecting relevant XML nodes to generate RDF triples from. We show that the time complexity of our mapping algorithm is linear in the size of the XML input and also prove its practical efficiency with an evaluation on large real-world data.

Categories and Subject Descriptors

I.7.2 [**Document Preparation**]: Markup languages; D.2.12 [**Interoperability**]: Data mapping

Keywords

XML, RDF, XML Filter

1. INTRODUCTION

XML documents [21] are widely being used to store and transfer information not only in business systems. XML-based languages are used to represent information in many domains and have gained particularly wide acceptance as standard data exchange formats in e-business. Examples of popular XML formats are *DatexII* for mobility, *XBRL* for financial data and *AutomationML* in the engineering domain.[1] XML documents can be queried using the *XQuery*[2] XML Query language, and transformed into text or differently structured XML files using *XML Stylesheet Language Transformations* (XSLT)[3]. They use *XPath* expressions[4] as a common subset for addressing content in XML documents.

Many companies and organizations are maintaining XML data, or expose legacy data, for example, from relational databases, as XML for data exchange purposes. Given the growing importance of Linked Data[5], exposing the data as RDF further improves its reusability. In particular, the world-wide unique URI/IRI identifiers of RDF facilitate unique identification and linking. Also, RDF data using different vocabularies is easier to integrate than XML documents using different schemas. The need to publish RDF data from XML sources has resulted in the development of a number of tools for mapping XML to RDF. Breitling [5] proposed a conversion from XML to RDF via XSLT[6]. It provides direct conversion without requiring an XML schema and human interaction. The converted RDF data contains XPath information for retaining the hierarchical information of the XML data source. However, this approach has three drawbacks. First, users must have knowledge of the relatively complicated XSLT transformation language if they want to define custom mapping rules. Second, additional effort and post-processing is required for replacing the subject URIs of the generated RDF documents if the XPath-style expressions are not deemed suitable as URIs. Third, stream processing of the data is not supported[7]. Other XSLT-based approaches [12, 13] show the same limitation w.r.t. stream processing and required XSLT knowledge. The query-based *XSPARQL* [1, 3] bridges XQuery and SPARQL to lift XML to RDF and lower RDF to XML. XSPARQL follows global-as-view approach using dynamic query translation to convert different data formats to RDF. However, users have to learn other languages in addition to SPARQL and XQuery. Again scalability is limited, since the XQuery engine processes data only in a non-streaming mode. This is due to the XSPARQL engine not being able to transform conventional query patterns into XQuery because the free version

SEMANTiCS '15, September 15-17, 2015, Vienna, Austria
© 2015 ACM. ISBN 978-1-4503-3462-4/15/09…$15.00
DOI: http://dx.doi.org/10.1145/2814864.2814880

[1]See http://www.datex2.eu, http://www.xbrl.org, and https://www.automationml.org.
[2]http://www.w3.org/TR/xquery-30/
[3]http://www.w3.org/TR/xslt-30/
[4]http://www.w3.org/TR/xpath-30/
[5]http://www.w3.org/standards/semanticweb/data
[6]XSLT 1.0. http://www.w3.org/TR/xslt/
[7]http://www.gac-grid.de/project-products/Software/XML2RDF.html

of Saxon XQuery engine[8] does not support data stream processing[9]. For many of the mentioned approaches, users need to have knowledge of XSLT, XSPARQL, XQuery, R2RML or SPARQL. On the other hand, their scalability is limited due to the lack of stream processing support or large data.

We propose a template language which uses RDF template patterns resembling RDF triple statements comprising simplified XPath expressions for generating subjects and objects. In order to easily select the relevant subjects and objects in XML which are expressed as XPath, we propose an RDF triple selection method to aggregate similar triples with the same template pattern. Additionally for reducing the size of the manipulated data and dealing with stream data, we introduce an efficient XML filtering technique. Our XML filtering technique uses a SAX parser to process XML data streams. Therefore, our approach can handle extremely large XML data and provides user friendly templates composed of RDF triple patterns including simplified XPath expressions.

The contributions of this paper are as follows:

1. **Intuitive mapping language.** We design a template language for mapping XML to RDF based on XPath expressions. The template describes the namespace of resources' URIs and the locations of subjects and objects in XML using XPath expressions.
2. **Smart interpretation of concise XPath expressions.** To enable low-effort authoring of mappings, we aim at concise XPath expressions. Because of their conciseness, these may end up being ambiguous. We resolve such ambiguities with a strategy to select those subjects and objects from the XML input that form the most plausible RDF triples. Our selection strategy is based on the distance from the Lowest Common Ancestor (LCA) of these elements to the closest nodes.
3. **Capability of processing data stream.** We introduce an XML filtering method using SAX to consume large data as streams.

Section 2 provides an overview of our approach. In section 3 we present the evaluation of our implementation. Section 4 presents related work. We conclude our implementation and introduce our future tasks in section 5.

2. APPROACH

After presenting our architecture, we present the template language, template parser, candidate nodes selectors and XML filtering respectively.

2.1 Architecture

As shown in Figure 1, our approach needs two input files: one template file and one XML file. For user defined templates, we developed a parser to retrieve triple patterns from the template file. The parsed triple patterns are fed to two core modules, the candidate nodes selector and the XML filter. The candidate nodes selector selects candidate nodes from the XML file satisfying the construction rules given in the template. Subsequently, the XML filter processes the XML document as a stream. If the filter finds matching XML nodes for the construction rules, these XML nodes will be used to create RDF subjects or objects according to the construction rules. The candidate selector will produce triples once it obtained sufficient nodes for generating triples.

[8]Saxon. http://www.saxonica.com

[9]Streaming in XQuery. http://bit.ly/1GAban8

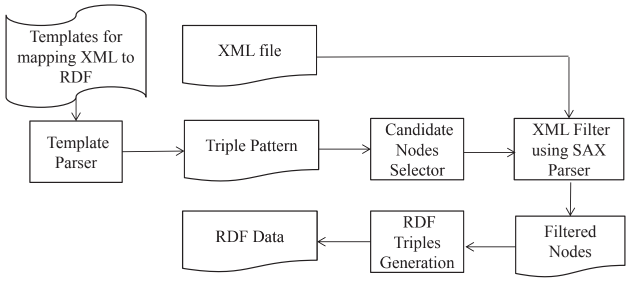

Figure 1: Architecture of Mapping XML to RDF

2.2 Template Language based on XPath Expressions

We assume that users have knowledge not only about the XPath language and its data model[10], but also about the RDF data model. In this paper, we denote RDF triples as $(s, p, o) \in R \times P \times R \cup L$. We call R the set of resources, P the set of properties and L the set of literals. We propose a template consisting of the following three parts: default namespace (abbreviated as **D**), namespace declaration (abbreviated as **N**) and construction of RDF triples (abbreviated as **C**):

D	defaultnamespace = URI-reference[11]
N	namespace *PrefixVAR* = URI-reference namespace *Prefix* = URI-reference namespace *Prefix* = URI-reference ...
C	*TriplePattern*

The syntax of *PrefixVAR* and *TriplePattern* is defined in Listing 1.

In the default namespace section, users indicate the URI of the default namespace. The key word `defaultnamespace` should appear only once. The format of the URI follows W3C's SPARQL specification. If not provided explicitly, the default namespace is defined as `http://www.w3.org/1999/02/22-rdf-syntax-ns#`. In the namespace section, users can define prefixes for URI namespaces. In contrast to the default namespace section, the number of prefix definitions is not limited.

The construction section contains triple patterns consisting of prefix, expression and type for subject, predicate and object. If no prefix is given, the default namespace is used. The expression contains a limited XPath expression which support a subset of XPath definitions. If the type is not specified, *resource* is assumed as default. For objects, considering `ResourceType` in the grammar, XSD types[12] can be used to specify the types of literals. Blank nodes can also be created; however, users have to define explicit blank node IDs for them.

Listing 2 shows an example template; Figure 2 shows an example XML document. In Figure 2, we use an XPath Data Model (XDM) tree (similar to a DOM tree) to present the XML document. We use subscript numbers to distin-

[10]XDM. http://www.w3.org/TR/xpath-datamodel/

[12]http://www.w3.org/TR/rdf11-concepts/#xsd-datatypes

Listing 1: The grammar definitions (PN_LOCAL refers to http://www.w3.org/TR/rdf-sparql-query/#rPN_LOCAL; VARNAME refers to http://www.w3.org/TR/rdf-sparql-query/#rVARNAME; QName refers to http://www.w3.org/TR/REC-xml-names/#NT-QName)

```
TriplePattern          ::= SubjectPattern PredicatePattern ObjectPattern
SubjectPattern         ::= (Prefix ',' SubjectExpression)| SubjectExpression|BlankNodePattern
PredicatePattern       ::= (Prefix ',' PredicateExpression)| PredicateExpression
ObjectPattern          ::= (Prefix ',' ObjectExpression ',' ResourceType) | (Prefix ',' ObjectExpression)
                         | (ObjectExpression, ResourceType) | ObjectExpression | BlankPattern
BlankNodePattern       ::= '_' ',' BlankNodeExpression
BlankNodeExpression    ::= LimitedXpathExpression | PN_LOCAL
Prefix                 ::= VARNAME
SubjectExpression      ::= LimitedXPathExpression | PN_LOCAL
PredicateExpression    ::= PN_LOCAL
ObjectExpression       ::=  LimitedXpathExpression | PN_LOCAL
LimitedXPathExpression ::= '/' RelativeLocationPath? '/text()' | '/' RelativeLocationPath?'/@' QName
RelativeLocationPath   ::= QName | RelativeLocationPath'/'QName
ResourceType           ::= 'string' | 'boolean' | 'decimal' | 'integer' | 'double' | 'float' | 'date' | 'time'
                         | 'dateTime' | 'dateTimeStamp' | 'gYear' | 'gMonth' | 'gYearMonth' | 'gMonthDay'
                         | 'duration' | 'yearMonthDuration' | 'dayTimeDuration' | 'byte' | 'short' | 'int'
                         | 'long' | 'unsignedByte' | 'unsignedShort' | 'unsignedInt' | 'unsignedLong'
                         | 'positiveInteger' | 'nonNegativeInteger' | 'negativeInteger' | 'nonPositiveInteger'
```

guish different XML nodes with the element or attribute names, we denote attribute nodes with a @ prefix, and use double quotes to denote the textual content of attributes and elements.

Listing 2: Example template

```
defaultnamespace = http://example.com/#
namespace ex = http://example.com/#
namespace foaf = http://xmlns.com/foaf/0.1/

ex,/result/person/@id foaf,name /result/person/name/text(),string
```

2.3 Template Parser

When one template is prepared, the parser parses the template according to the following steps.

1. Finding default namespace: fetches and validates the default namespace URI using a regular expression.
2. Fetching user-defined namespace: fetches and validates user-defined namespace bindings using a regular expression.
3. Fetching construction rules: fetches and validates and prefixes, types and the corresponding expressions for the subjects and predicates and objects of RDF triples.

Finally, the parser will output the triple patterns to RDF Triple Selection (in subsection 2.4).

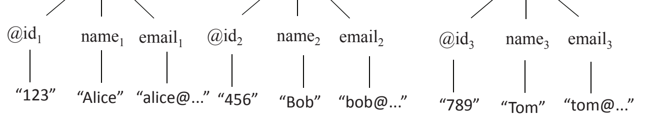

Figure 2: Tree representation of an example XML document.

2.4 Candidate Nodes Selector

The method for selecting candidate nodes from the XML document for the generation of RDF triples is one of our core modules. Selecting relevant nodes for creating subjects and objects in a certain triple pattern requires a distance measure between XML node candidates for generating subject and object.

Considering the template in Listing 2 and the example data in Figure 2, the simplified candidate triple results are shown in the following table:

No.	Subject	Predicate	Object
1	123	foaf:name	”Alice”
2	123	foaf:name	”Bob”
3	123	foaf:name	”Tom”
4	456	foaf:name	”Alice”
5	456	foaf:name	”Bob”
6	456	foaf:name	”Tom”
7	789	foaf:name	”Alice”
8	789	foaf:name	”Bob”
9	789	foaf:name	”Tom”

From these candidates, the finally generated RDF triples should be the triples in lines 1, 5 and 9. All the others are not relevant, since the nodes for generating the respective subjects and objects are not contained in the same sub-tree of the DOM. To select nodes in the same sub-tree of the DOM for generating RDF triples, an intuitive way is to measure the distance between the corresponding XML nodes. Before explaining our distance measurement for selecting candidate nodes, we introduce a basic distance measure between XML nodes. Afterwards, we introduce our candidate nodes selector, which is inspired by the SLCA-based approach [24].

For a given node `n` in the DOM tree, the distance of this node from the root node is the number of edges from root to `n`. In the DOM tree, we call the nodes satisfying an XPath expression related to a subject pattern *subject nodes* and the nodes satisfying an XPath expression related to an object pattern *object nodes*. Then we call a node `a` an ancestor node of node `n` if `a` is on the path from the root node to `n` and `n` is called a descendant node of `a`. The distance between the ancestor node `a` and `n` is the number of edges from `a` to `n`, denoted by `anc_dist(a,n)`. The distance between the root node `root` and `n` is consequently `anc_dist(root,n)`. For n_1 and n_2, a node n_c is called a common ancestor node, denoted by `com_anc`(n_1,n_2) if n_c is an ancestor node of n_1 and n_2. The distance of n_c to n_1 and n_2 is the minimum value of `anc_dist`(n_c,n_1) and `anc_dist`(n_c,n_2), denoted by `com_dist`(n_c,n_1,n_2). Given a set `C` of common ancestor of nodes n_1 and n_2, we call the node n_L the *lowest common ancestor* (LCA) node if n_L has the minimum value of `com_dist`(n_i,n_1,n_2) of all $n_i \in C$. Obviously, n_L has the highest distance from the root node of all nodes in `C`. We denote the distance between two nodes n_1 and n_2 as `com_dist`(n_L,n_1,n_2).

Now, we use the above distance definition to define the distance based on the XPath expressions. We assume that there are XPath expressions for the subject pattern P_S and the object pattern P_O, the grammar of which is defined in subsection 2.2. For a subject node n_s derived from P_S and an object node n_o derived from P_O, the lowest common ancestor n_L of n_s and n_o can be expressed by the intersection of P_S and P_O. Given our restricted definition of XPath expressions, the XPath of the LCA node n_L of two distinct DOM nodes is a substring of the corresponding XPath expressions P_S and P_O for the two nodes. It works because the corresponding XPaths are absolute paths.

Therefore, the XPath of the LCA node can be found by comparing the element names of the two XPath expressions. After the XPath expression n_L is found, the corresponding lowest common ancestor can help us decide on the closest subject and object nodes.

Considering Figure 2, according to the XPath expressions in Listing 2, the LCAs' XPath is `/result/person` and the corresponding LCAs are $person_1$, $person_2$ and $person_3$. For processing the input, Depth-First Search (DFS) can be used. The selected LCAs can be identified prior to their descendant nodes during the DFS. While finding one matching LCA, the matched descendant nodes with corresponding XPath expressions can be selected for composing triples. For $person_1$, the matched nodes are id_1 and $name_1$, then the triple in line 1 of our result table is generated. The triples in lines 5 and 9 will also be generated as required.

If developers want to create their own selector for ancestors, they can extend our software implementation[13] to create new selectors.

2.5 XML Filtering

XML filtering is another core module of our implementation. To filter out relevant subjects and objects in one triple pattern and process data streams, we integrate the *YFilter*[7] XML filter approach with the above mentioned selector and templates.[14] After loading the template and pre-calculating the corresponding LCAs' XPath expressions, the XML filter performs by Algorithm 1. In lines 7–19 and lines 30–36, since the LCA is ancestor for all subject and ob-

[13]https://github.com/allen501pc/XML2RDF

[14]We developed our own implementation of the YFilter approach, since the reference implementation was complicated to extend and integrate.

Algorithm 1 The filtering algorithm

```
Input: XML input, $xml$, XPath expressions for subject patterns $Path_s$, object patterns $Path_o$ and pre-computed XPath expressions $Path_l$ for the LCAs.
Output: The RDF triples.
BEGIN
 1: Build a YFilter with the XPath expressions $Path_s$, $Path_o$ and $Path_l$.
 2: Declare two lists, $List_{subject}$ and $List_{object}$.
 3: Use the YFilter to load $xml$ using the SAX parser.
 4: for each event reported by SAX do
 5:     Fetch the current element $e$
 6:     if the event happens on start tag then
 7:         Load $e$ into YFilter's element stack
 8:         Fetch the XPath expression $Path_e$ from $e$
 9:         if $Path_e$ matches $Path_l$ and $e$'s attributes are accepted by YFilter then
10:             Fetch the accepted attribute $attrset$
11:             for each $attr \in attrset$ do
12:                 if $attr$ matches $Path_s$ then
13:                     $List_{subject}.add(attr)$
14:                 end if
15:                 if $attr$ matches $Path_o$ then
16:                     $List_{object}.add(e)$
17:                 end if
18:             end for
19:         end if
20:     end if
21:     if the event happens on end tag then
22:         Fetch the XPath expression $Path_e$ from $e$
23:         if $Path_e$ matches $Path_l$ then
24:             Generate corresponding RDF triples from $List_{subject}$ and $List_{object}$.
25:             Clear all data in $List_{subject}$ and $List_{object}$.
26:         end if
27:         Pop out $e$ from YFilter's stack.
28:     end if
29:     if the event happens on context then
30:         Fetch the XPath of $e$'s context, $Path_{text}$
31:         if $Path_{text}$ matches $Path_s$ then
32:             $List_{subject}.add(text)$
33:         end if
34:         if $Path_{text}$ matches $Path_o$ then
35:             $List_{object}.add(text)$
36:         end if
37:     end if
38: end for
END
```

ject nodes and the SAX parser is processing depth-first, it is easy to make decisions for storing corresponding subject and object nodes. Subsequently, we can generate corresponding RDF triples and clear the subject and object lists in lines 24–25.

3. EVALUATION

3.1 Theoretical Complexity

For an XML input with `k` XML elements and `n` mapping rules, the main overhead of the mapping procedure is executing the XML filter; the filter does $\mathcal{O}(n)$ rule mappings for each XML element and it processes $\mathcal{O}(k)$ elements. Thus, the time complexity is $\mathcal{O}(k * n)$.

3.2 Practical Evaluation Setup

We conducted our evaluation on a physical machine with Windows 7, 64-bit platform, an Intel i5-2450M CPU, 8 GB RAM and a 7200 rpm HDD. We used Java 1.7 to run the implementation, with the default Java heap size. We choose two large-scale, real-world datasets as XML input: *OpenAIRE*[15] and *DBLP*[16].

The OpenAIRE (Open Access Infrastructure for Research in Europe) project manages scientific publications and associated scientific material via repository networks. It aggregates Open Access publications and links them to research data and funding bodies. The OpenAIRE project provides data query APIs for users.[17]. The APIs return XML data by default. The OpenAIRE data model comprises five entity types: (1) result (of a research project, e.g., a publication or a dataset), (2) person, (3) organization, (4) data source and (5) project. The corresponding XML schema is implemented in XSD.[18]

Due to limitations of network access and security policies, we did not obtain the full data for this experiment. We selected 500,000 XML records provided by the data manager of OpenAIRE. The selected records are separately exported to 10 files and the size of each is around 300 MB. In this experiment, 95 construction rules[19] for the five entity types mentioned above are used to generate RDF triples.

The other XML dataset from DBLP comprises entities of the following types: (1) books, (2) thesis, (3) papers in proceedings, (4) journal articles, (5) reference works. The size of this dataset is around 1.4 GB. In the experiment for the DBLP dataset, 34 construction rules are used to generate RDF triples.[20]. The construction rules are divided into three parts which users are most interested in: (1) thesis, (2) in-proceedings and (3) journal. The numbers of construction rules for thesis, journal and in-proceedings are 10, 8, 16 respectively. The experiments are executed separately for the three entity types to inspect the effect on the number of construction rules and matched triples. The default output data format for these experiments is in N-Triples.

[15]OpenAIRE. https://www.openaire.eu/

[16]DBLP. (Access Time: December 2, 2014) http://dblp.dagstuhl.de/xml/

[17]http://api.openaire.eu/

[18]OpenAIRE XSD. https://www.openaire.eu/schema/0.2/doc/oaf-0.2.html

[19]https://github.com/allen501pc/XML2RDF/tree/master/XML2RDF/templates3

[20]https://github.com/allen501pc/XML2RDF/blob/master/XML2RDF/dblp_new.x2r

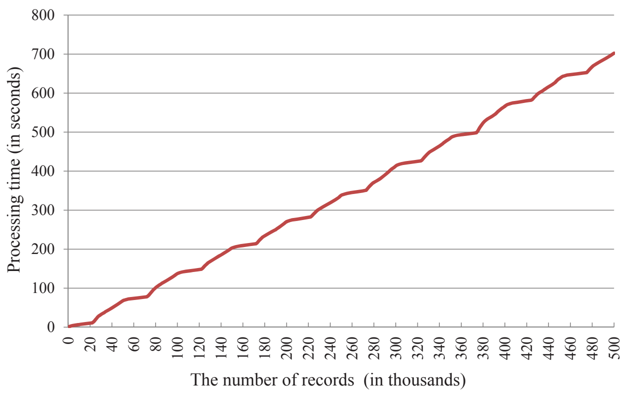

Figure 3: Accumulated Processing Time for the OpenAIRE Dataset

The accumulated processing time for the OpenAIRE dataset is shown in Figure 3. We accumulated the processing time while executing the mapping procedure for each XML document. The processing time grows approximately linearly with the number of records. This is due to the XML filter processing the data using the SAX parser, which works in a streaming mode. The result shows that our method can process very large data. The overall processing time is approximately 11 minutes for 3 GB of data, mainly due to relatively slow disk access. The memory usage is 670 MB.

The other experiment result with the DBLP dataset is shown in Table 1. For *thesis*, the number of rules is larger than *journal* but the execution time is lower because there are fewer matching triples in *thesis* than *journal.* For *journal* and *proceeding*, the results show that the execution time increase as well as the number of construction rules increases. The memory usage for *thesis*, *journal* and *proceeding* are 350 MB, 538 MB, 570 MB respectively.

In the OpenAIRE setting, we compared the performance and maintainability of our XML→RDF approach to mappings from an HBase NoSQL database and from CSV [19][21]. The results, detailed in [19], prove that our XML→RDF approach works on large datasets. However, it needs more computation time than the mapping from HBase because it does not employ a parallel processing strategy. Also, with regard to maintainability, the comparison to high-level languages for mapping CSV to RDF points out further potential for improving *our* mapping language.

3.3 Limitations of the Template Language

We perform an approach with the capability of processing stream data processing and offer a template language for users to write concise mapping rules to map XML to RDF. Remaining limitations of this language are:

1. **Understanding XML Schema** Our language requires absolute XPath expressions. Thus, the candidate nodes selector can select the region of related candidate XML nodes according to the pre-calculated XPath expressions. This means that users may not be able to write the suitable XPath expressions without knowing the XML schema used.

[21]See the project page for details: http://eis.iai.uni-bonn.de/Projects/OpenAIRE2020.html

Table 1: Experimental results for the DBLP dataset.

Data type	# of rules	running time (in seconds)	# of generated triples	size of generated triples
Thesis	10	35.85	34,825	2 MB
Journal	8	121.7	7,242,768	473 MB
Proceeding	16	206.42	13,845,502	817 MB

2. **Restricted Expression of Language** The XPath expressions of our template language are a subset of all XPath expressions; our language does not provide any string or mathematical operations. For XSLT-based approaches, the XSLT provides such operations to perform mappings from XML to RDF. If the memory is large enough, the XSLT-based approach may provide flexible operations than our proposed approach.

Despite these, our provided language can help users who may not know complex query languages perform mapping tasks from XML to RDF.

4. RELATED WORK

To map XML to RDF and efficiently transform the XML data, a number of approaches were proposed.

To efficiently filter XML data, XML filtering technology is widely adopted. *YFilter* [7] proposed an *NFA*(Non-deterministic Finite Automaton)-based representation of XPath expressions which combines all XPath queries in a *single* machine. It merges the common sub-paths of XPath queries, which then results in great improvements of structure matching performance during processing the filtering procedure. Because it uses NFA to model the filter, users who are familiar with NFA can easily create their own filters. Although there are several approaches [15] [23] improving YFilter, we still choose YFilter as our filtering approach in order to ensure better maintainability. For speeding up the execution of XPath queries on XML data, an indexing technology [22] can cache the XPath queries and thus reduce the query time. For converting XML data to RDF and when creating a number of mappings at once, the indexing scheme is not suitable for our approach.

Semantic Annotations for WSDL and XML Schema (SAWSDL) [10] annotates the semantic attributes of WSDL components presented in XML format. Afterwards, it uses XSLT to convert XML into RDF and make an inverse transformation employing XSLT and SPARQL. *Gleaning Resource Descriptions from Dialects of Languages* (GRDDL) [6] also uses XSLT to transform XML data into RDF. Breitling implemented a direct, schema-independent transformation, which retains the XML structure [5]. However, converting this generic RDF representation into a domain-specific one requires post-processing on the RDF side, e.g., transformations using SPARQL CONSTRUCT queries. On the other hand, the current version[22] of this approach is based on XSLT 1.0, which does not support data stream processing. Thuy *et al.* proposed an XML Schema-based scheme [18] for mapping XML to RDF. It derives RDF Schema from XSD then produces the corresponding RDF from the XML. In its experiment, the proposed scheme could be passed by W3C's RDF validator. However, its generations of identifiers and predicates may not be flexible for domain experts. Klein's approach used RDF Schema to specify what XML elements and attributes should be extracted and what RDF classes and properties they should be mapped to [11]. Furthermore, it does not automatically interpret the nesting relation between two elements in an XML as a named property between resources.

Polleres *et al.* [1][3] proposed `XSPARQL` language to bridge XQuery and SPARQL to transform XML to RDF and vice versa. *XSPARQL* proposes a dynamic query translation to convert different data formats to RDF. This approach provides a language rewriting engine to transform `XSPARQL` into `XQuery`. Afterwards, it uses an `XQuery` engine to construct RDF. However, `XSPARQL` is not easy to understand by those users who have no knowledge of SPARQL-like query language. A subset of XQuery[9] is suitable for streaming input, but it is neither supported by the `XSPARQL` implementation nor by the free version of the Saxon XQuery processor required to run XSPARQL. However, the user interface provided by [3] forced users writing query texts, which may be complicated for novice users. The visual editor of XSPARQL *XSPARQL–Viz* [9] provided a web user interface. The GUI proposes a *drag* and *drop* visual query editor to design their mappings between XML and RDF. In order to create such GUI, it utilizes the Spring Inversion of Con(IoC) framework to build layered services which include generations of: (1) XML schema, (2) RDF schema, (3) query and (4) result. For creating mappings between XML and RDF, users have to provide *URI* of the input or upload an XML or RDF dataset. Subsequently, the schema generation service will generate corresponding schema for the input. Users can use its graphical query editor to edit queries and indicate the output format. Then the corresponding queries will be generated by the query generation service. Once the queries are generated, the result generation service will execute corresponding *XSPARQL* and visualize the output results. For query based approaches, the *SPARQL2XQuery* [16, 2] system has the similar drawbacks as XSPARQL – knowledge of SPARQL and OWL are needed. Also, as an query answering system for users it may cause large query processing time to construct the corresponding RDF or OWL data. Domain experts have to use the *XS2OWL* (XML Schema to OWL) model in order to create accurate mapping rules for the existing XML schema. The XML to RDF mapping tool – *Krextor* [12] provides an easy way to let users develop mapping rules for converting XML to RDF. However, knowledge of XSLT is needed with this approach.

In the *DTD2OWL* [17] approach, the mapping rules were defined by an OWL ontology and DTD[23]. Hence, this approach requires knowledge of DTD and OWL. Also, it is not flexible with regard to identifier generation for resources. The identifier generation automatically creates identifiers by concatenating each element name in XML and an incremental numbers. Considering mapping XML to a high

[22]http://www.gac-grid.de/project-products/Software/XML2RDF.html

[23]http://www.w3.org/XML/1998/06/xmlspec-report-19980910.htm

Table 2: Comparison of mapping and transformation methods for XML to RDF/OWL.

Method	Input	Mapping direction	Output	Language or grammar expression	Data stream processing
SAWSDL [10]	XML	$\leftrightarrow$	RDF	WSDL,XSLT	No
GRDDL [6]	XML	$\rightarrow$	RDF	GRDDL,XSLT	No
Breitling [5]	XML	$\leftrightarrow$	RDF	XSLT	No
Thuy [18]	XML	$\rightarrow$	RDF	XSLT	No
Klein [11]	XML	$\rightarrow$	RDF	XSLT,RDFS	No
XSPARQL [1][3]	XML/RDF	$\leftrightarrow$	XML/RDF	XSPARQL,XQuery,SPARQL	XQuery engine req.
XSPARQL–Viz [9]	XML/RDF	$\leftrightarrow$	XML/RDF	XSPARQL,XQuery,SPARQL	XQuery engine req.
SPARQL2XQuery [16] [2]	XML/RDF	$\rightarrow$	RDF/OWL	OWL,SPARQL	XQuery engine req.
Krextor [12]	XML	$\leftrightarrow$	RDF	XSLT	No
XML2OWL [4]	XML	$\rightarrow$	OWL	XSLT	No
DTD2OWL [17]	XML/DTD	$\rightarrow$	OWL	DTD, OWL	No
JXML2OWL [13]	XML	$\rightarrow$	RDF	XPath,OWL	No
Davy [20]	XML	$\rightarrow$	RDF	OWL	No
RML [8]	XML,JSON,RDB	$\rightarrow$	RDF	RML, R2RML	Implementation req.

level language, OWL, *XML2OWL* [4] is a native method for mapping XML to OWL using XSLT stylesheets. However, post-processing is needed for generating a domain-specific and aligned OWL data.

JXML2OWL [13] automatically converts the XML data to OWL. This approach utilizes XPath expressions to compose mapping rules to create OWL data. To create the mappings, an OWL ontology and the corresponding mapping rules have to be provided in advance and this causes complicated preliminary work for users. To alleviate this, an user interface to assist users to define rules is provided. However, the approach causes heavy computation while processing mappings.

Davy *et al.* [20] propose a rule based approach to extract RDF instances of an OWL ontology from XML input. For creating conditional mapping rules, this approach integrates `SPARQL` queries to select data from the existing OWL ontology. Thus, both knowledge of `OWL` and `SPARQL` are required. Most legacy data is usually stored in relational databases (RDB). For mapping the data in a relational database into RDF, a many tools are implemented [14]. For creating such mappings, *R2RML*(RDB to RDF Mapping Language) is recommended by W3C[24]. *RML*(RDF Mapping Language) [8], a superset of *R2RML* mapping language, supports the definition of mappings from variant data sources to RDF. The features of this approach are: (1) be able to process streaming data, (2) a extension language of *R2RML*, (3) support of variant data sources such as relation database, XML and JSON(JavaScript Object Notation). However, the drawbacks of this approach are: (1) users have to learn knowledge of *R2RML*, *RML*, (2) the mapping rules are complicated and (3) the XPaths of each parent node of the subject and object nodes in XML should be given by users; in many cases for creating one triple, users have to write several lines of mapping rules[25]. Due to the above reason, it is difficult to create the mapping rules by the users who are not familiar with *RML* and *R2RML*.

Finally, we provide a comparison table in Table 2 for the above mentioned methods. To summarize, existing approaches for mapping XML to RDF, users are often forced to understand both of XML, OWL and RDF data models. Thus such approaches may not be very suitable for those users who have only knowledge of XPath and RDF triples. Also, most of existing approaches are not be able to process data streams, which is especially important for the use cases where processing large amounts of data is required.

5. CONCLUSIONS AND FUTURE WORK

In this paper, we propose an efficient approach to map XML to RDF which has an ability to process data streams. For creating mapping rules, we design a template language for mapping XML to RDF based on XPath. For those users who have little knowledge of XPath and RDF triples, the template approach lets them easily create their own mapping rules. To select candidate triples, we employ a measure to calculate the distance between related subjects and objects in XML and then generate corresponding RDF triples. As can be seen from the experiment results, our implementation performs well on large data. On the other hand, developers can extend our proposed approach from https://github.com/allen501pc/XML2RDF to create different template language parsers, XML filters and candidate nodes selectors.

In the future, we aim to develop different XML filters and candidate nodes selectors for different requirements. For processing big data, integrating our approach with MapReduce platforms such as Hadoop[26] and Spark[27] is also a promising research direction.

Acknowledgments

This work is partly supported by the European Commission under FP7 grant 601043 and H2020 grant 643410. This work was performed while the first author, Jyun-Yao Huang, was visiting the University of Bonn under the grant No. 103-2911-I-005-503 by Taiwan MOST/DAAD Sandwich program.

References

[1] W. Akhtar et al. *XSPARQL: Traveling between the XML and RDF worlds–and avoiding the XSLT pilgrimage.* Springer, 2008.

[24] http://www.w3.org/TR/r2rml/

[25] See lines 56–94 in http://bit.ly/1BaQL5K

[26] https://hadoop.apache.org/

[27] https://spark.apache.org/

[2] N. Bikakis et al. “The XML and Semantic Web Worlds: Technologies, Interoperability and Integration: A Survey of the State of the Art”. English. In: *Semantic Hyper/Multimedia Adaptation*. Ed. by I. E. Anagnostopoulos et al. Vol. 418. Studies in Computational Intelligence. Springer Berlin Heidelberg, 2013, pp. 319–360. URL: http://dx.doi.org/10.1007/978-3-642-28977-4_12.

[3] S. Bischof et al. “Mapping between RDF and XML with XSPARQL”. English. In: *Journal on Data Semantics* 1.3 (2012), pp. 147–185. URL: http://dx.doi.org/10.1007/s13740-012-0008-7.

[4] H. Bohring and S. Auer. “Mapping XML to OWL Ontologies”. In: *Leipziger Informatik-Tage* 72 (2005), pp. 147–156.

[5] F Breitling. “A standard transformation from XML to RDF via XSLT”. In: *Astronomische Nachrichten* 330.7 (2009), pp. 755–760.

[6] D. Connolly. *Gleaning Resource Descriptions from Dialects of Languages (GRDDL)*. W3C, 2007. URL: http://www.w3.org/TR/grddl/.

[7] Y. Diao et al. “Yfilter: Efficient and scalable filtering of XML documents”. In: *Data Engineering, 2002. Proceedings. 18th International Conference on.* IEEE. 2002, pp. 341–342.

[8] A. Dimou et al. “Mapping Hierarchical Sources into RDF Using the RML Mapping Language”. In: *Semantic Computing (ICSC), 2014 IEEE International Conference on.* 2014, pp. 151–158.

[9] S. Z. H. Gillani, M. I. Ali, and A. Mileo. “XSPARQL-Viz: A Mashup-Based Visual Query Editor for XSPARQL”. In: *The Semantic Web: ESWC 2013 Satellite Events*. Springer, 2013, pp. 219–224.

[10] H. L. Joel Farrell. *Semantic Annotations for WSDL and XML Schema*. W3C, 2007.

[11] M. Klein. “Interpreting XML documents via an RDF schema ontology”. In: *Database and Expert Systems Applications, 2002. Proceedings. 13th International Workshop on.* 2002, pp. 889–893.

[12] C. Lange. “Krextor–an extensible XML to RDF extraction framework”. In: *Scripting and Development for the Semantic Web (SFSW)* (2009).

[13] T. Rodrigues, P. Rosa, and J. Cardoso. “Moving from syntactic to semantic organizations using JXML2OWL”. In: *Computers in Industry* 59.8 (2008), pp. 808 –819. URL: http://www.sciencedirect.com/science/article/pii/S016636150800064X.

[14] S. S. Sahoo et al. “A survey of current approaches for mapping of relational databases to RDF”. In: *W3C RDB2RDF Incubator Group Report* (2009).

[15] P. Saxena and R. Kamal. “System architecture and effect of depth of query on XML document filtering using PFilter”. In: *Contemporary Computing (IC3), 2013 Sixth International Conference on.* 2013, pp. 192–195.

[16] I. Stavrakantonakis et al. “Sparql2xquery 2.0: Supporting semantic-based queries over xml data”. In: *Semantic Media Adaptation and Personalization (SMAP), 2010 5th International Workshop on.* IEEE. 2010, pp. 76–84.

[17] P. T. T. Thuy, Y.-K. Lee, and S. Lee. “DTD2OWL: Automatic Transforming XML Documents into OWL Ontology”. In: *Proceedings of the 2Nd International Conference on Interaction Sciences: Information Technology, Culture and Human*. ICIS ’09. Seoul, Korea: ACM, 2009, pp. 125–131. URL: http://doi.acm.org/10.1145/1655925.1655949.

[18] P. T. T. Thuy, Y.-K. Lee, and S. Lee. “XSD2RDFS and XML2RDF Transformation: a Semantic Approach”. In: *The Second International Conference on Emerging Database (EDB 2010), Jeju, Korea*. 2010.

[19] S. Vahdati et al. “Mapping Large Scale Research Metadata to Linked Data: A Performance Comparison of HBase, CSV and XML”. In: *Metadata and Semantics Research*. CCIS. Springer, 2015. arXiv: 1506.04006 `[cs.DB]`.

[20] D. Van Deursen et al. “XML to RDF Conversion: A Generic Approach”. In: *Automated solutions for Cross Media Content and Multi-channel Distribution, 2008. AXMEDIS ’08. International Conference on.* 2008, pp. 138–144.

[21] W3C. *Extensible Markup Language (XML)*. World Wide Web. URL: http://www.w3.org/XML/.

[22] B. Zhang and Z. Zhuang. “Efficient Structural XML Index for Multiple Queries”. In: *Recent Advances in Computer Science and Information Engineering*. Springer, 2012, pp. 423–431.

[23] H. Zhao, W. Xia, and J. Zhao. “The Research on XML Filtering Model using Lazy DFA”. In: *Journal of Software* 7.8 (2012), pp. 1759–1766.

[24] M. Zhou, H. Hu, and M. Zhou. “Searching XML data by SLCA on a MapReduce cluster”. In: *Universal Communication Symposium (IUCS), 2010 4th International*. IEEE. 2010, pp. 84–89.