Trinity College Dublin
Coláiste na Tríonóide, Baile Átha Cliath
The University of Dublin

TOWARDS AUTONOMIC UPLIFT OF INFORMATION

Anuj Singh

A Dissertation submitted to the University of Dublin, Trinity College

in fulfillment of the requirements for the degree of

Master of Science in Computer Science

(Mobile and Ubiquitous Computing)

2015

Declaration

I, the undersigned, declare that the work described in this dissertation is, except where otherwise stated, entirely my own work and has not been submitted as an exercise for a degree at this or any other university.

Anuj Singh

Dated: August 31, 2016

Permission to Lend and/or Copy

I, the undersigned, agree that Trinity College Library may lend or copy this thesis upon request.

Anuj Singh

Dated: August 31, 2016

Acknowledgement

I would first like to thank my supervisor, Prof. Declan O'Sullivan for proposing this topic. Prof Declan O'Sullivan provided me all the required support and guidance throughout duration of the project. I would like to thank Dr. Christophe Debruyne, a research fellow from the knowledge and data engineering group for suggesting the improvements in the design. I also would like to thank my family and friends for their love and support during this year. Finally I would like to thank Mobile and ubiquitous computing class for making this a memorable year.

Anuj Singh

University of Dublin, Trinty College

August 2016

TOWARDS AUTONOMIC UPLIFT OF INFORMATION

Anuj Singh

Supervisor: Declan O’Sullivan

In this project we propose an approach to transform the XML data into RDF using XQuery. The mappings from XML to RDF are encoded in XQuery. We are also transforming the XQuery to RDF, allowing us to analyze, manipulate and recompose mappings automatically. The project is motivated as a way to provide a robust mechanism to support the maintenance of the mappings automatically, particularly to cope up with frequent change in data schema or ontology.

The developed prototype for this research is based on the MarkLogic Server. The prototype uses an XQuery parser [33] for parsing the XQuery. It provides two ways to analyze and manipulate the mappings 1) GUI- where the user manually updates the mappings 2) SPARQL- triples of XQuery can be automatically updated using SPARQL queries. We evaluated the system for both functionality and usability. We applied the developed solution to the industry problem of transforming the metadata to RDF. We transformed the Trinity digital collections resource metadata from FileMaker to MODS RDF. For evaluating the system, we changed the mappings (FileMaker to MODS RDF) automatically as well as manually to see the changes in the final output. We evaluated the usability with the help of multiple users, who updated and analyzed the mappings using the prototype. Users rated the prototype based on their experience of operating it.

Overall, we found that our prototype successfully uplifted both the XML and XQuery. Based on the evaluation it can be said that the prototype was successful in analyzing, manipulating and recomposing the mappings automatically.

Table of Contents

List of Tables 9
List of Figures 10
Chapter 1 11
Introduction 11
1.1 Motivation 11
1.2 Dissertation goal 13
1.2.1 Research question 13
1.2.2 Research objective 13
1.3 Research methodology 14
1.4 Dissertation outline 15
Chapter 2 17
Background 17
2.1 Introduction 17
2.2 Semantic Web and its growth 17
2.3 Building blocks of Semantic Web 18
2.4 RDF Serialization formats 20
2.4.1 RDF/ XML 20
2.4.2 Turtle 21
2.5 SPARQL 21
2.6 Lifting and Lowering of Information 22
2.7 Summary 22
Chapter 3 23
State of the art 23
3.1 Introduction 23
3.2 Mapping formats 23
3.2.1 RML 24
3.2.2 EDOAL 26
3.2.3 D2R Map 28
3.3 Query based approach 29
3.3.1 XSPARQL 30

3.3.2 XQueryX 32
3.4 Summary 33
Chapter 4 36
Design 36
4.1 Introduction 36
4.2 Requirements 36
4.2.1 Functional requirements 36
4.2.2 Non-Functional requirements 38
4.3 Technical architecture 38
4.3.1 Component description 39
4.3.2 Flow of operations 41
4.4 Design Challenges and Resolutions 42
4.4.1 How to transform XQuery in an RDF data model automatically? 42
4.4.2 How to transform “tree structure of XQuery” to XML? 43
4.4.3 XML should be uplifted to which RDF serialization format? 44
4.4.4 How to treat blank nodes in the uplifted XML? 45
4.5 Summary 45
Chapter 5 46
Implementation 46
5.1 Tools and Technologies 46
5.2 Conventions for implementation 47
5.3 Use cases implementation description 49
5.3.1 Use case 1: XQuery to RDF 49
5.3.2 Use case 2: Input XML to RDF 53
5.3.3 Use case 3: Rendering mappings in GUI 53
5.3.4 Use case 4: Analysis and manipulation of triples 56
5.3.5 Use case 5: Switching to new dataset 58
5.4 Summary 59
Chapter 6 61
Evaluation 61
6.1 Evaluation: Test cases 61
6.1.1 Test case 1: Uplifting input XML to RDF 62

6.1.2 Test case 2: Update mapping through user GUI-I 63
6.1.3 Test case 3: Update mapping through user GUI-II 64
6.1.4 Test case 4: Update mapping through user GUI-III 65
6.1.5 Test case 5: Update mapping through user GUI-IV 66
6.1.6 Test case 6: Changing mapping by updating triples-I 68
6.1.7 Test case 7: Changing mapping by updating triples-II 69
6.1.8 Test case 8: Preserve blank node 70
6.1.9 Test case 9: Analyze the transformed RDF triples-I 71
6.1.10 Test case 10: Analyze the transformed RDF triples-II 72
6.1.11 Test case 11: Changing mapping by updating triples-III 73
6.1.12 Test case 12: Changing mapping by updating triples-IV 74
6.2 Evaluation: User perspective 75
6.3 Summary 78
Chapter 7 79
Case Studies 79
7.1 Introduction 79
7.2 Trinity library digital collection metadata 79
7.3 Employee dataset 82
Chapter 8 84
Conclusion 84
8.1 Project overview 84
8.2 Contribution 86
8.3 Future work 86
8.4 Final remark 87
References 88
Appendix A 95
Appendix B 99

List of Tables

Table 1: Mapping approaches 35

List of Figures

Figure 1: Web Layer Cake 18
Figure 2: Triple instance 19
Figure 3: RDF/XML instance 21
Figure 4: Turtle instance 21
Figure 5: SPARQL Query instance 21
Figure 6: Lifting and lowering of data 22
Figure 7: Mapping process with and without RML 25
Figure 8: Alignment instance[25] 27
Figure 9: D2R mapping process[10] 29
Figure 10: Schematic view of XSPARQL[29] 31
Figure 11: XSPARQL implementation architecture[29] 31
Figure 12: Technical architecture of proposed system 39
Figure 13: Selection of “Title” and “AttributedArtist” from the input XML for mapping 48
Figure 14: Corresponding RDF structure for “Title” element of input XML 49
Figure 15: A sample XQuery for instance 50
Figure 16: Tree structure of sample XQuery 50
Figure 17: Intermediate XML structure of sample XQuery 51
Figure 18: RDF structure of sample XQuery 52
Figure 19: Execution of transformation of input XML to RDF 53
Figure 20: Interface to view/ update mappings manually 54
Figure 21: Input element “last_name” that has to be mapped 55
Figure 22: mapping rule (RDF structure) for Input element “last_name” 56
Figure 23: Interface to execute SPARQL queries 57
Figure 24: Execution of SPARQL queries 57
Figure 25: code snippet to update "XQuery RDF" based on triple’s changes 58
Figure 26: Interface to upload updated XQuery 58
Figure 27: code to replace the existing XQuery with updated one 59
Figure 28: Ease of completing the task 76
Figure 29: Time taken to complete the task 77
Figure 30: Support information 77
Figure 31: Title information of a resource in FileMaker XML 81
Figure 32: Title information of a resource in MODS RDF 81
Figure 33: mapping of city in triple maps 83
Figure 34: Mapping of city in XQuery 83

Chapter 1

Introduction

1.1 Motivation

Over the last decade the information available on the Web has become more complex [3]. Organisations, institutes and enterprises (from hospitals to entertainment) often model their data independently and hence data integration and translation have become a major problem for organizations which are interested in utilizing the data on Web [34]. Linked data is a way to integrate data of various domains by using ontologies, Uniform Resource Identifiers (URIs) and Resource Description Framework (RDF) [35]. To remove the restriction posed by the different data models, organisations have started lifting their data to RDF, which has not only given them the standard data model, but also allowed them to publish their data in the linked data fashion such that it can be integrated easily [36].

With the adoption of Semantic Web and linked data, a challenge for extraction of data from various data sources remains as the data on the Web is of different quality and format which makes it difficult to consolidate [37]. There are solutions available which perform the mapping of one format to another to perform the extraction and transformation. However, in case any of the formats gets changed, then these mappings need to be established again and further the transformation engine has to be adjusted accordingly [38].

Digital repositories are something in today's world which are accessed by almost everyone. These contain resources from a wide domain (journals, books, software and other resources) but these repositories have the lack of structured and robust ways to interoperate between related resources across various domains [39]. A wide domain of resources add more complexity in the process of extraction and transformation of data, thus it requires a robust way by which the mappings and transformation engine can be maintained automatically.

Another use which motivates this research is the concern of digital libraries about handling the heterogeneous metadata. Furthermore, the library community has been searching for a solution of archiving and version control, due to the difference in the format of the metadata of the resources [49]. As metadata is an important part of any application, it is imperative to keep it in best representation. This gives rise to a need of an approach by which metadata can not only be transformed easily, but also the mapping should be easily maintainable to cope up with further changes.

So far researchers have only considered areas like generic solutions for mapping which can be used to express or define mappings from heterogeneous data sources to RDF. [10], [20] and [28] are the most recent advancement in the field of mapping which suggests various approaches and the mapping languages for carrying out the transformation of data to RDF. However, in the current state of the art, negligible attention has been given to the representation and structure of the mapping. This offers us an area to research where we can determine the representation of the mapping such that this representation can be used for ongoing analysis and re-composition of mappings automatically.

The crux of this research is to represent mappings in the RDF format. Going forward the maintenance of the mappings could be very challenging and cumbersome task, especially when the format of any end data source tends to change very often. RDF is a machine understandable language [40] and by representing the mappings in RDF one can analyze and manipulate them automatically and thus make the maintenance of these mappings less challenging.

1.2 Dissertation goal

1.2.1 Research question

The research question addressed in this thesis is:

"To what extent the mappings encoded in XQuery can be represented using Resource Description Framework (RDF) such that it allows performing ongoing analysis and re-composition of mappings automatically?"

1.2.2 Research objective

We propose a solution for the representation of the mappings encoded in XQuery using RDF so that it can be analyzed, manipulated and recomposed automatically. In particular, we intend to:

- Understand the current state of the art for mapping of data and their integration, it will take into account the whole maintenance life cycle of the mapping.
- Understand the challenges faced by the digital libraries in order to handle the metadata present in a variety of formats.
- Develop a prototype which will transform XQuery to RDF document.
- Estimate the precision and recall of the transformation of XQuery to RDF automatically.
- Develop an interface which will allow the user to update the mappings stored as XQuery RDF.

- Transform the XML (format 1) to RDF (format 2) document based on the mapping (established in XQuery based RDF).

- Estimate the precision and recall of the transformation of XML to RDF using the mapping (established in XQuery based RDF).

- Analyze the output from the transformation of XQuery to RDF using SPARQL.

- Analyze the output from the transformation of XML to RDF based on the mapping (established in XQuery based RDF) using SPARQL

- Develop the library use case as a means to evaluate the prototype of the approach developed.

1.3 Research methodology

Our proposed approach introduces four additional components in the conventional processes [15] and [29] of transforming the data to RDF using predefined set of mappings.

- Mapping container: An XQuery to establish the base mappings.

- XQuery Parser: It acts as a transformer for converting the XQuery expression to the XML based tree structure.

- RDF generator: It converts the XML based tree structure to RDF.

- RoundTrip Engine: It generates the XQuery expression from the RDF.

We propose an approach to represent the XQuery encoded mappings in RDF such that it can be recomposed and analyzed automatically. We also present a prototype for the proposed approach using the MarkLogic content server [41]. We then evaluate the prototype of the approach using following two dataset to be uplifted.

- The FileMaker data for expressing the metadata of library resources. We have gathered this data from the TCD library digital collection. The TCD library suggested that FileMaker XML needs to be uplifted into MODS RDF. We have also gathered all the necessary mapping information from the TCD library end. The sample FileMaker dataset can found in the attached CD (\lifting\assets\Filemaker\DRIS-SAMPLE-Filemaker.xml)

- The Employee dataset for expressing the details about the employees. We have gathered this data from "Northwind database". This data needs to be uplifted into FOAF. The mapping information of employee dataset to FOAF has been provided by Dr. Christophe Debruyne. The sample employee dataset can found in the attached CD (\lifting\assets\Filemaker\employees.xml)

1.4 Dissertation outline

Chapter 2: This chapter provides an insight to the information which is required to understand the background of the semantic and linked data. It highlights the growth of the Semantic Web and what are the building blocks of Semantic Web. It also focuses on the concept of uplift of data.

Chapter 3: This chapter outlines the current state of the art approaches in the area of semantic mapping. Section 3.2 describes the various mapping formats which are used in the mapping of RDB, XML and ontologies. In later sections the focus is on the query based approaches which are XSPARQL and XQueryX.

Chapter 4: This chapter describes the design of the project. It captures the requirements of the project, which are functional and nonfunctional. In the later section it presents the technical architecture, the challenges faced during the designing process and what are the countermeasures we took to overcome those challenges.

Chapter 5: This chapter illustrates the details about the implementation of the project. The implementation is based on the technical architecture specified in the section 4.3. This chapter also captures the details about the tools and technologies used in this implementation.

Chapter 6: This chapter describes the evaluation of the developed prototype. It covers the evaluation of the system in two areas: 1) Functionality of the system and 2) System usability evaluation.

Chapter 7: In this chapter, we explore the application of the developed prototype on some industrial problems. We discuss the uplift of “Trinity library digital collection metadata” and “Employee dataset”.

Chapter 8: This chapter describes the summary of the research and our findings. It also captures the detail about the future work to be employed in this research.

Chapter 2

Background

2.1 Introduction

This chapter outlines the background of the Semantic Web and linked data. The Semantic Web and its growth have been described in the next part of this chapter. Third part of this chapter describes the building blocks of the Semantic Web; it also specifies the various serialization formats of RDF. This chapter ends with the introduction of lifting and lowering of information in the world of Semantic Web.

2.2 Semantic Web and its growth

[1]The conventional way to publish data on the Web is through CSV, or XML or in HTML tables. This Web is called as a conventional hypertext Web. In conventional hypertext Web the relationship between the two documents is not explicit, this is basically due to the data format which is HTML and it suffers from the limited vocabulary which cannot state link between entities in different documents. But in recent years the Web has evolved as a global information space where both the documents and data can be linked. This evolution was the practice of appropriate rules to publish data which in result produce the effect of Linked data. Practicing these principles of Linked data has led to the extension of Web to global information space which connects the different domain of the industries such as humans, infrastructure, biological data, chemical data, geographic data, community's data and scientific data. With the help of linked data user can actually traverse into different domain when it starts from browsing some specific domain.

[2]The origin of the Web was considered as an information store where the information can be used for human-human communication as well as the machines can also play a vital role to make this communication more efficient. One of the major challenges to this

is that information on the Web is not properly structured for machine browsing. [2] It also provides an idea about the basic assertion model. The idea behind this model is to have a generic data structure so that any prospective application can communicate to that structure. [2] It specifies that the generic data model which can serve this purpose is Resource description framework.

Figure 1: Web Layer Cakeshows the architecture of the Semantic Web. This architecture is also called the Semantic Web Stack or Semantic Web Cake or Semantic Web Layer Cake. This structure also asserts that the Semantic Web is an extension of the classical Hypertext Web and not a replacement.

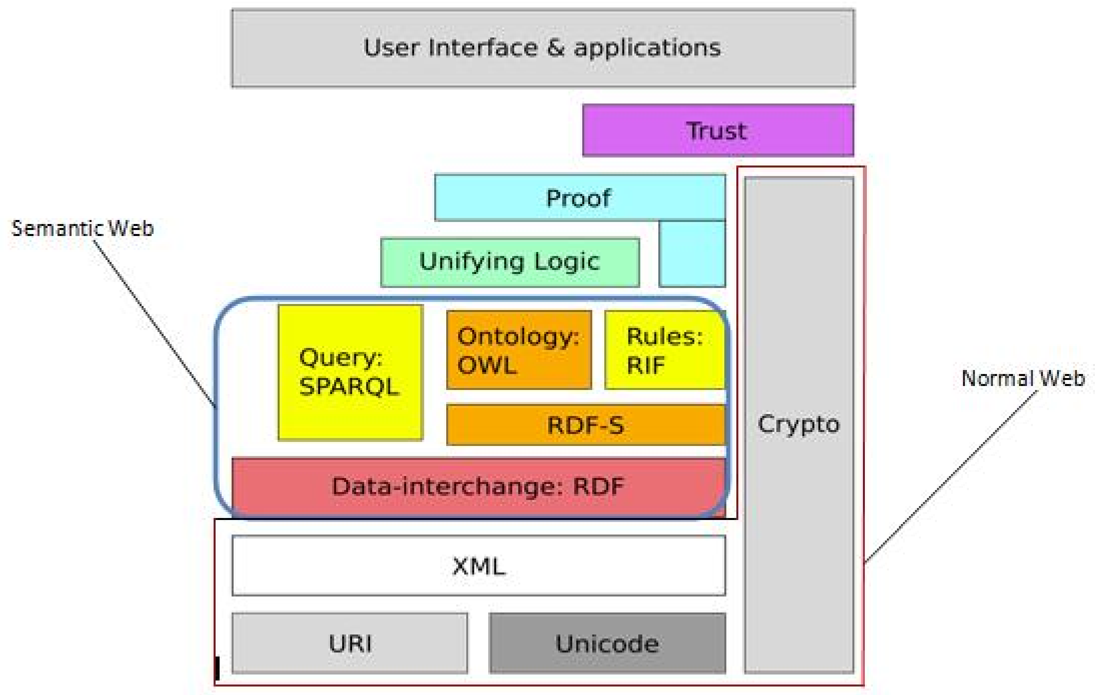


Figure 1: Web Layer Cake

2.3 Building blocks of Semantic Web

The communication between machines is completely based on the data, for every request and response the machines exchange the data. It suggests that in order to get the interoperability over the data the software and hardware should be able to access and interpret the data [3]. Resource Description Framework (RDF) is the data model which can be used as a generic data structure for the communication between various applications or programs [2].

W3C specifies that RDF is the standard data structure for exchange of information over the Web [4]. RDF does not replace the linking structure of existing Web but it extends it by using the URIs. In place of the names of relationships between the resources, it also uses the URIs for the resources.

RDF describes a resource with the help of statements; these statements are in the form of Subject-Predicate-Object expression. These expressions are known as triples. In the triples subject is the resource about which the triple is describing, predicate defines the property of the resource, and object is the value of the property which could be a literal value or it can redirect to the subject of another resource [5].

Figure 2 depicts the role of subject, predicate and object in the triple by providing a simple example of the industry based triple, where *T-shirt* is the resource, *color* is the property of the T-shirt and *white* is the value of the property of the T-shirt.

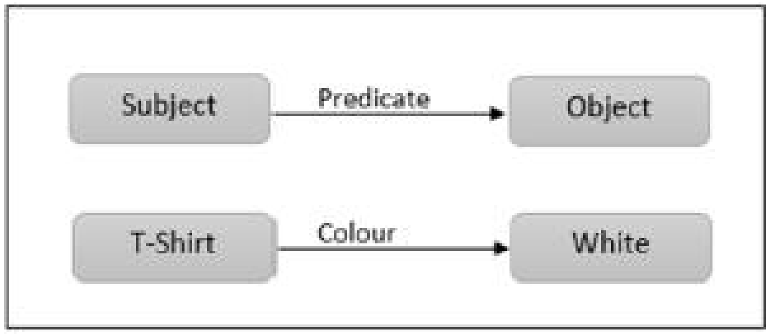


Figure 2: Triple instance

2.4 RDF Serialization formats

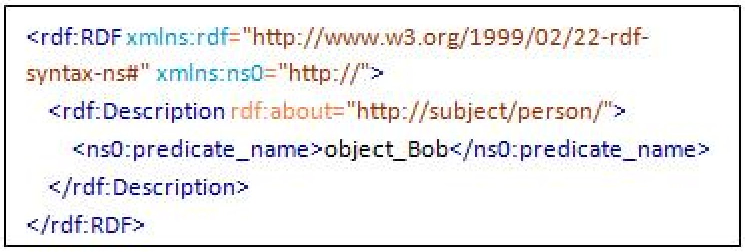
There are multiple serialization formats which can be used for the description of the resource, currently below are the serialization format which are used in the industry:

- RDF/ XML
- Turtle
- N-Triples
- JSON-LD
- N-Quads

Among these RDF/ XML and Turtle are the most popular and widely used serialization formats.

2.4.1 RDF/ XML

XML syntax representation for RDF is known as RDF/ XML. In order to represent the RDF in XML the subject, predicate and object of the triples need to be represented using the XML term [6]. Nodes are in RDF/ XML are basically the URI references or the literals, in the case of blank node, it needs to have an identifier. This identifier is local to the document and it should be a non-RDF URI references.

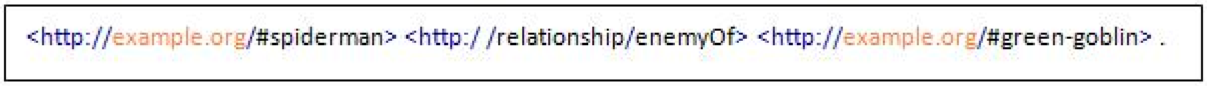


```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#" xmlns:ns0="http://">
  <rdf:Description rdf:about="http://subject/person/">
    <ns0:predicate_name>object_Bob</ns0:predicate_name>
  </rdf:Description>
</rdf:RDF>
```

Figure 3: RDF/XML instance

2.4.2 Turtle

This representation of RDF is very compact and human readable as this is represented by the natural text form. Triple in this can be expressed very easily as it is just a sequence of subject, predicate and object separated by whitespace and terminated by the full stop [7].

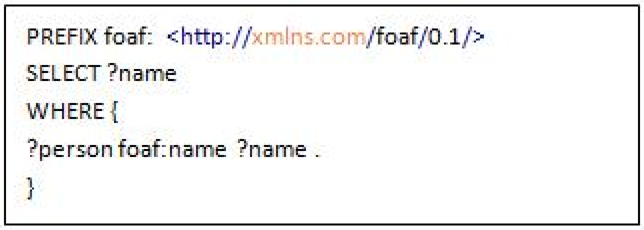


```
<http://example.org/#spiderman> <http:/ /relationship/enemyOf> <http://example.org/#green-goblin> .
```

Figure 4: Turtle instance

2.5 SPARQL

SPARQL is the query language for the RDF graphs, SPARQL query needs a triple pattern which is known as basic graph pattern [8]**.** SPARQL query contains two parts, first is "SELECT" clause which is used to construct the result, second part is "WHERE" clause used the triple pattern provided to match against the RDF provided.

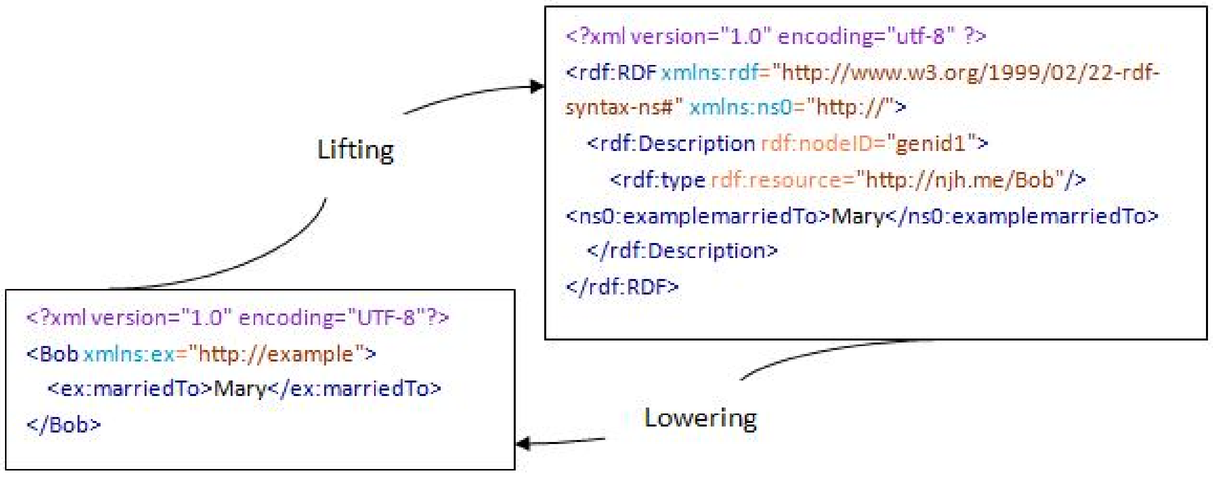


```
PREFIX foaf:  <http://xmlns.com/foaf/0.1/>
SELECT ?name
WHERE {
?person foaf:name  ?name .
}
```

Figure 5: SPARQL Query instance

2.6 Lifting and Lowering of Information

The process in which the native representation of the data has been transformed into its semantic representation is called as uplift of data [9]. For an instance XML is the native representation and its semantic representation is the RDF and OWL. The reverse process which bringing the data back into its native state is called as the lowering of data.



```
<?xml version="1.0" encoding="utf-8" ?>
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-
syntax-ns#" xmlns:ns0="http://">
  <rdf:Description rdf:nodeID="genid1">
    <rdf:type rdf:resource="http://njh.me/Bob"/>
<ns0:examplemarriedTo>Mary</ns0:examplemarriedTo>
  </rdf:Description>
</rdf:RDF>
```

```
<?xml version="1.0" encoding="UTF-8"?>
<Bob xmlns:ex="http://example">
  <ex:marriedTo>Mary</ex:marriedTo>
</Bob>
```

Figure 6: Lifting and lowering of data

2.7 Summary

This chapter contains the details about the background of the Semantic Web and linked data along with the information of its building block. It also has the information about the different formats of RDF. It also describes the concept of lifting and lowering in Semantic Web which is used in this research extensively. Hence this can be served as an introduction to the state of the art.

Chapter 3

State of the art

3.1 Introduction

The goal of the Semantic Web is to provide the meaning to the data which is available on the Web. The process of transforming the data from its native representation to its semantic representation is known as lifting of information [10]. The Semantic Web is not the replacement of, but the extension of the existing Web. In Semantic Web the information available on the Web can be shared among the machines and people [11]. The Semantic Web allows the programs, devices and even household appliances to produce and use the data on the Web [11]. This research requires having an understanding of the current and previous researches which have been done in the area of mapping and lifting of information. [10], [12], [13], [14], [15], [17] and [18] proposed the generic solutions for lifting the data but a very little attention has been given to the representation of the mapping.

The idea behind this research is to represent XQuery encoded mappings in RDF. As RDF is the semantic representation of the data, these mappings can be analyzed, manipulated and even recomposed automatically.

This chapter will outline the current researches in the area of semantic mapping. It also presents the query based approach for mapping as this research is based on the query based approach.

3.2 Mapping formats

In today's world machine needs to exchange data from various sources. To understand the exchanged data semantic mapping needs to be done among the different data formats. One of the conventional and well known schemes for sharing and storing the data is XML. XML is ubiquitously used by the database developers and programmer for representing information, the advantage for using the XML is that it provides the additional flexibility when someone tries to structure the information in XML [26].

Another popular format to store the data is relational database [10]. According to [27] mapping between RDB and RDF can be represented in XSLT as XPaths. The actual implementation of the mapping can be either extract, Transform and Load (ETL) or a query-driven implementation. Queries used for mapping process could be either SPARQL or it can be translated to equivalent SQL queries. SPARQL queries can be executed on the RDF structure and SQL queries can be executed on the relational data model.

Many proofs of concepts have been created for the generic solution for expressing mappings. Some of the solutions involved relational database schema and ontologies to form mapping while some used “table to class” and “column to predicate” approach to generate mappings. As survey suggests that there is no standard way of mappings between RDB to RDF. FOL, XPath expressions, and tool-specific languages can be used for representation of mapping [27].

In this research an XQuery based approach will be used to encode the mapping. The objective is to determine the representation of the mappings such that these can be analyzed, manipulated and recomposed automatically.

3.2.1 RML

RDF mapping language is a generic structure which can be scaled according to the need. RML is basically used to lift the data in various formats to the RDF [19]. RML is the superset of the W3C standard R2RML which uses the features of the R2RML and also have some additional functionality.

R2RML is specific in defining the mapping for the relation database to RDF data structure [16]. The mapping is based on the triple maps which specify how the triples will going to be generated [16], [19]. Triple maps are responsible for defining the rules for zero or more RDF triples, it contains three major components which are 1) Logical table 2) Subject map and 3) Predicate-Object maps [19]. Subject map is used to define the rule which generates the URIs for the resource and these URIs are used as the subject of all the RDF triples. Predicate-Object map is responsible for defining the rule which creates the predicate for all RDF triple and object map is responsible for creating the mapping for the object of all RDF triples. Logical table is used to store all the mappings [16].

RML retains the R2RML's definition of mapping while it removes the database specifications. The biggest difference between the RML and R2RML is that the input source in R2RML is limited to only one database while in RML the input source could be one or more relational databases **[16]**.

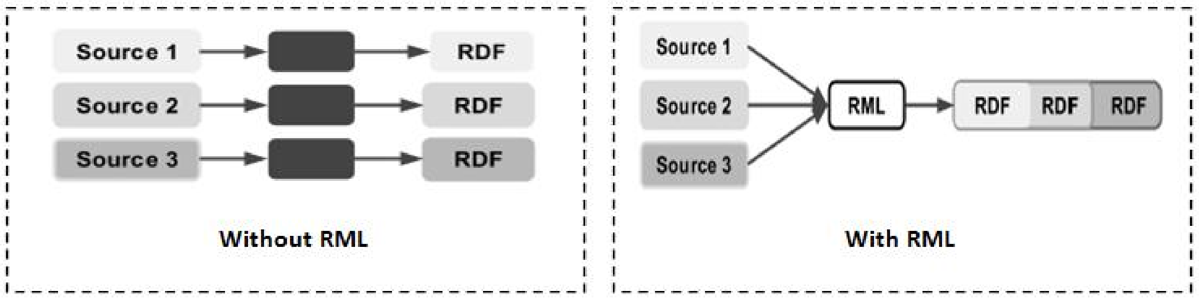


Figure 7: Mapping process with and without RML

RML does not have its unique syntaxes it uses the syntax of R2RML. RML requires the valid input data source and the mapping definitions to be mapped to RDF. As RML is dealing with multiple formats, some details need to be specified before executing the mappings which are *logical source, reference formulation, iterator, logical reference, referencing object map.* Logical source is the extension of R2RML logical table and is used to identify the source of input data, reference formulation is used to deal with the different serialization of data, iterator is used to specify pattern based on which the input data needs to be extracted, logical reference is a property for the reference and its value must be valid and needs to be present in the input source, referencing object map is the “join condition child reference” which refers to the parent of the data extracted [16].

RML model can be implemented using two approaches which are "mapping driven", "data driven". In mapping driven the processor executes triple maps consecutively. In a data driven model the process drives by the extractor module, the processor extracts the triple maps based on the patterns specified for iteration [16].

[20] Evaluated the RML mapping against the Datalift tool [21]. The Authors used the same RML mapping for ten input files which is completely opposite than the Datalift where the mapping needs to be redefined for all ten files. The authors suggested that the solution using the RML was optimal because of the semantic richness and better interlinking of the output.

RML is able to combine the triple maps from heterogeneous data sources to generate the triples. The maintenance of the mapping is a semi-automatic process and the user should have an understanding of the languages like XML, JSON and XPATH. In my opinion RML is not suitable for handling complex mappings which involve complex XPATH conditions. It would be really clumsy to depict the XPATH condition which involves the use of axes/ predicate with RML relationship's features. Our research is not based on RML because of its semi-automatic nature and it is difficult to handle the complex mappings in RML.

3.2.2 EDOAL

Ontology is one of the most efficient ways to achieve collaboration among the systems, which are using data in various formats. Sometimes ontologies are not compatible with each other. In order to make them compatible with each other it requires some re-engineering of these ontologies. Re-engineering is completely based on identifying the correspondence between the two ontologies [22]. Process of Identifying the correspondence between the entities of two ontologies is called as "ontology alignment" [23].

The Expressive and Declarative Ontology Alignment Language (EDOAL Language) extends the alignment format in order to get more precision in the correspondence of entities of two ontologies. This can be achieved by expressing the entities in the alignment format with the help of certain constructor and operator [22]. EDOAL is majorly based on the OWL which is why it uses many features of OWL, but EDOAL is not completely dependent on the OWL and the reason for not adopting OWL completely is to not restrict EDOAL to a particular language [22]. EDOAL uses the constraints provided by description logic or OWL, it also allows the basic named entities to be grouped with the help of Boolean operators. EDOAL is not restricted to the specific entity names it also have a pattern language which allow to express patterns for the entities which can be abstract [24].

In my opinion, the major drawback of the EDOAL is the user involvement. Correspondence made at design end needs to be finalized by the human. In application environment, users are generally not ontology specialist. Inspection of the correspondence needs the user to be the domain expert. Generally the task related to alignments is easy for humans, but it is fairly complex if we involve machines in place of humans. As it involves human involvement at a very high scale, we prefer to select an alternative approach for this research.

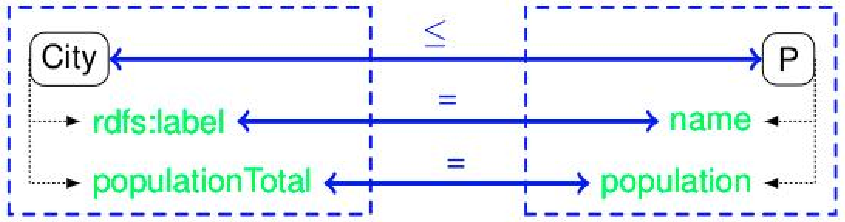


Figure 8: Alignment instance[25]

3.2.3 D2R Map

One of the ways to express the mapping between relational databases and RDF is through D2R map, it is an XML based language [10]. As XML allow flexibility in structuring the information so to take this advantage this language has been designed. The reason why it needs to be flexible is that it needs to handle complex relational structure without changing the schema [10]. The SQL statements are directly used in the mapping rules and the result set will be later combined and mapped to create the instances.

The D2R map process has following four steps [10]:

- Records set needs to be selected from the input source.
- Record set needs to be grouped by the columns of the specific class map.
- The URI needs to be assigned after creating the class instance, if it's a blank node, then blank node identifier needs to be assigned.
- Properties for instances need to be created according to the data type and object property bridge.

In my opinion D2R map is the suitable way to handle complex mapping of relational database to RDF. Its XML mapping format grants the structural flexibility while the provision to include SQL statements directly in the mapping allows establishing complex mappings. Another positive side of D2R map is that it does not involve user interference in that way it is less prone to error. But our research is not inspired by this approach as the selected format of the input data is XML for lifting.

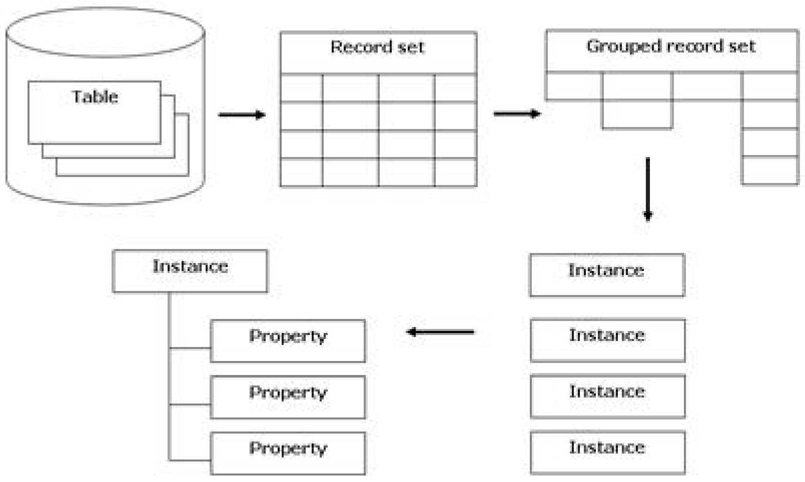


Figure 9: D2R mapping process [10]

3.3 Query based approach

Data integration is a problem which almost every information system is dealing with [32]. The process which is responsible for maintaining the information system is ETL (Extract Transform Load). The data integration problem becomes more complicated when the data present in the different sources adheres to the different formats. To deal with these type of situation an extended ETL process is required which transform every format to a single format while extraction [32].

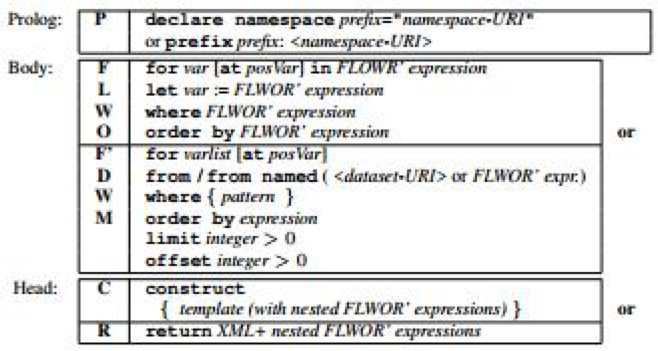
[30] Suggested the query based mapping approaches which are XSPARQL as it is the combination of XQuery and SPARQL, the motive behind using XSPARQL is that it can query XML as well as RDF in the same framework. Details about the XSPARQL have been provided in section 3.3.1.

Another query based mapping approach is XQueryX, the biggest part of this research is influenced by XQueryX but we are not using XQueryX format completely. The XQueryX is the XML representation of XQuery, and it is machine understandable which can be useful in an environment where the program needs to query the code for analysis or other purposes [32]. Details about the XQueryX have been provided in section 3.3.2.

3.3.1 XSPARQL

Existing ways of transformation of documents solely rely on XSLT. [30] suggests some ways to overcome the errors which can be caused solely by relying on the XSLT for transformation, it proposes a technique which is based on XQuery and SPARQL for such transformation, these transformations can be performed by merging XQuery and SPARQL into a new language XSPARQL. XSPARQL provides a way of mapping between XML and RDF in both directions. In Web of data service community applications needs to communicate with XML and RDF data model thus it needs a transformation mechanism [30].

XSPARQL allows you to query Extensible Markup Language (XML) data structure and Resource description framework (RDF) structure in the same framework [29]. The benefit of the combination of two languages is that by the feature of SPARQL, we can access RDF and use Turtle to construct RDF graphs while having the feature of XQuery will allow us to access and manipulate the XML data structure [29]. In the body XSPARQL uses the SPARQL F'DWM block instead of XQuery FLOW block. The new F' (for) clause allow us to have multiple variables separated through whitespace. In the result construction step it allows us to directly create the RDF graph by using C (construct) clause instead of XQuery return clause. DWM clauses are responsible for organising the result such as putting a constraint thought a pattern and sorting the result through various parameters [29].



Prolog:	P	**declare namespace** *prefix*="*namespace-URI*" or **prefix** *prefix*: <*namespace-URI*>	
Body:	F	**for** *var* [**at** *posVar*] **in** *FLOWR' expression*	
	L	**let** *var* := *FLWOR' expression*	
	W	**where** *FLWOR' expression*	
	O	**order by** *FLWOR' expression*	or
	F'	**for** *varlist* [**at** *posVar*]	
	D	**from / from named** (<*dataset-URI*> or *FLWOR' expr.*)	
	W	**where** { *pattern* }	
	M	**order by** *expression*	
		limit *integer* > 0	
		offset *integer* > 0	
Head:	C	**construct** { *template (with nested FLWOR' expressions)* }	or
	R	**return** *XML+ nested FLWOR' expressions*	

Figure 10: Schematic view of XSPARQL [29]

The most updated version of XSPARQL is XSPARQL 1.1, it has all the features of SPARQL 1.1 and it is also compatible with JSON documents [28]. [28] Suggests that the XSPARQL is capable enough to handle the production system as it is currently being used extracting the social data from social media such as Facebook, LinkedIn, Google+ etc.

In my opinion the XSPARQL is a very powerful language if we have the information present in XML as well as RDF format. Its unique nature of executing the XQuery code as well as the SPARQL code allows the user to meet its requirement in the same framework. The concept of XSPARQL inspired this research to replace the sole use of XSLT for the purpose of lifting information. This research is using the XQuery for the lifting purpose. XSPARQL is not employed in this research because we are lifting the information in RDF/ XML format from XML. XSPARQL will be extremely useful where the information is being uplifted in turtle format from XML.

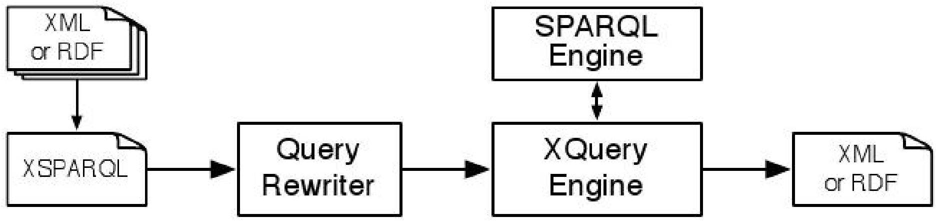


Figure 11: XSPARQL implementation architecture [29]

3.3.2 XQueryX

The XQueryX is the XML representation of XQuery, but it's not pretty well known. The XQuery statements will get parsed according to the W3C standards. It allows you to create an XML data structure of the XQuery. Later this XML structure will be transformed with the help of XSLT 2.0 [31] to get the final XQueryX format. xq2xml is the first application which provides us the mechanism to transform XQuery to XQueryX [31]. These syntaxes have been designed especially for the machines such that the they can interact with the XQuery and manipulate it, XQueryX is machine oriented, that is why it is not suitable for humans that is humans cannot conveniently read and write it [32].

[32] Indicated following environment where XQueryX could be useful.

- Parser reuse: In multiple systems if an XQuery needs to be executed, then a parser can create the XQueryX of the XQuery and then that XQueryX can be used by every system.

- Queries on Queries: As XQueryX is an XML representation of XQuery, so with the help of this XML structure we can query the XQuery such that if we want to query which 'for' clause is responsible for extracting the result from the specific XML, we can easily do these kind of reasoning over XQueryX.

- Generating Queries: In some XML environments it is more convenient to handle XQueryX format instead of the XQuery expression because the XQueryX can be handled by using the normal XML tool.

- Embedding XQuery into XML: If there is any need to import XQuery into XML, it is always good to import it in XML structure otherwise we cannot reason/ query over the XQuery portion.

[31] suggest that to perform another transformation over the XML representation of XQuery, the input one should choose is XML generated by the XQuery parser not the final XML generated by xq2xml tool. The output generated by the xq2xml tool is more complex and might be losing some information which could be useful to the creator of XQuery. According to [31] if an XQuery is transformed to XQueryX and when this XQueryX transformed back to XQuery it might add additional information to the XPath which might not be useful.

The latest version of the XQueryX schema and the XSLT (which is used to transform XQuery to XQueryX) is available at http://www.w3.org/2005/XQueryX/xqueryx.xsd and http://www.w3.org/2005/XQueryX/xqueryx.xsl respectively.

This research is heavily inspired by the XML representation of the XQuery. This XML can be later uplifted to RDF. As RDF is the machine understandable format, it can be used for analyzing, manipulation of XQuery automatically. In this study, we are using “XQuery parser” instead of “xq2xml”, one can download the xgrammar.zip from the https://www.w3.org/2005/qt-applets/xqueryApplet.html.

3.4 Summary

This state of the art outlined the various approaches which are currently in the practice for the mapping of native structure of data to its semantic structure. It has been compiled that very limited attention has been given to the structure of the mapping, the focus of the approaches is more towards the creation of generic language for the transformation.

This chapter described the various mapping approaches which can be used for the transformation of data to the RDF. Following are the descriptions all the approaches:

- RDF Mapping Language (RML): It is a generic mapping language which is used to define mapping rules from various data format to RDF.

- Expressive and Declarative Ontology Alignment Language (EDOAL): It is a language which is used to represent the correspondence between the entities of different ontologies.

- D2R map: It is an XML based mapping language. The SQL statements are directly used in the mapping rules and the result set will be later combined and mapped to create the instances.

- XSPARQL: It is another way of defining the mapping between XML and RDF, It is able to query XML and RDF both in the same framework as XSPARQL is the combination of XQuery and SPARQL.

- XQueryX: XQueryX is the XML representation of XQuery. XQueryX syntaxes have been designed especially for the interoperability of machines such that the machine can interact with the XQuery and manipulate it.

Approach	Input Sources	Mapping Format	Features	Maintenance	Comments
RML	Relational database, CSV, TSV, XML and JSON	Triple maps in triple format.	Triple maps from the heterogeneous input sources can be combined to generate the output triples.	Semi-automatic	RML generates default mappings but a customization is needed from the mapping author. [53] It suggests a semi-automatic approach to quickly generate the RML mappings using an extension built for Karma.
EDOAL	Ontologies	OWL	Identifying correspondence related tasks are comparatively easy for humans and there are several visually supported alignment tools.	Semi-automatic	Domain experts participate to improve the quality of the mappings, identified during the design.
D2R	Relational databases	XML	It allows flexibility by using XML based mappings and employing the SQL statements directly into the mapping such that complex relational structures can be mapped easily.	Automatic	Changes in the mapping can be detected automatically with the help of its d2r:autoReloadMappingFeature.
XSPARQL	RDF, XML	XSPARQL	Its unique nature of executing the XQuery code as well as the SPARQL code allows the user to meet its requirement in the same framework.	Automatic	For web service communication with RDF-client, it can perform lifting and lowering of data automatically.
xq2xml	XQuery	Vocabulary specifications	“xq2xml” is the first application which provides us the mechanism to transform XQuery to its XML representation.	Automatic	It takes valid XQuery as input so it does not require any changes as it works on the already specified vocabulary specifications.

Table 1: Mapping approaches

Chapter 4

Design

4.1 Introduction

This chapter describes the design of the project. This chapter covers the functional and nonfunctional requirements of the system. Later sections of this chapter describe the technical architecture of the proposed system. It also presents the details about the challenges faced during the design phase of the project, and then states the measures to overcome those challenges.

4.2 Requirements

This section elaborates the functional and nonfunctional requirements of the system. The main objective of this system is to uplift the XML data and represent the mappings in RDF. As RDF is machine understandable format, the mappings can be analyzed, manipulated and recomposed automatically. XQuery is used in this system for the purpose of transformation of XML to RDF. XML data from the following two different domains has been selected to uplift.

- The FileMaker dataset for expressing the metadata of library resources.
- The Employee dataset for expressing the details about the employees.

4.2.1 Functional requirements

This section describes the requirements implemented in the proposed system. The proposed system covers the following functionality:

- Security: User should be able to log in the system only with the valid credentials.

- Input data: System is compatible with the XML structure data, so the input should always be in XML format.

- Transformer: A base XQuery needs to be in place which contains the mapping of the input XML to RDF.

- Uplift input XML: Use the base XQuery to transform the input XML to RDF automatically.

- XQuery parser: The base XQuery, which contains the mapping to RDF needs to be transformed into RDF automatically.

- Round-trip engine: The RDF structure of the XQuery needs to be converted back into XQuery expressions.

- GUI to view mappings: Render the mappings on html page from the RDF structure of base XQuery.

- Update mappings: Rendered mappings should be able get updated and the changes should be reflected in the RDF structure of base XQuery.

- Create triples from XQuery: Store triples in the MarkLogic triple store from the RDF structure of base XQuery. Triples of “XQuery RDF” need to be stored in the graph whose base URI is “http://xquery/XQrdf”.

- Create triples from uplifted data: Store triples in the MarkLogic triple store from the RDF structure of input XML. Triples of “RDF of input XML” need to be stored in the graph whose base URI is “http://data.test.tcd.ie/resource”.

- Access triples: User should be able to read and update triples in the MarkLogic triple store through SPARQL query. Changes done by the SPARQL update query should be reflected in the RDF structure of base XQuery.

4.2.2 Non-Functional requirements

- The user should have an experience of XML and XQuery in order to update the mapping from GUI.

- The user should have an experience of SPARQL and triples in order to update the mapping through SPARQL query.

- The user manual should be in place such that users can update mappings and uplift the XML to RDF without any hitch.

- The system should not do anything with data except what is specified in the mapping.

- Uplifted RDF produced by the system should be a valid one.

- The response time of the system should be minimized.

4.3 Technical architecture

In order to attain the objective of this research a multi phase process has been adopted.

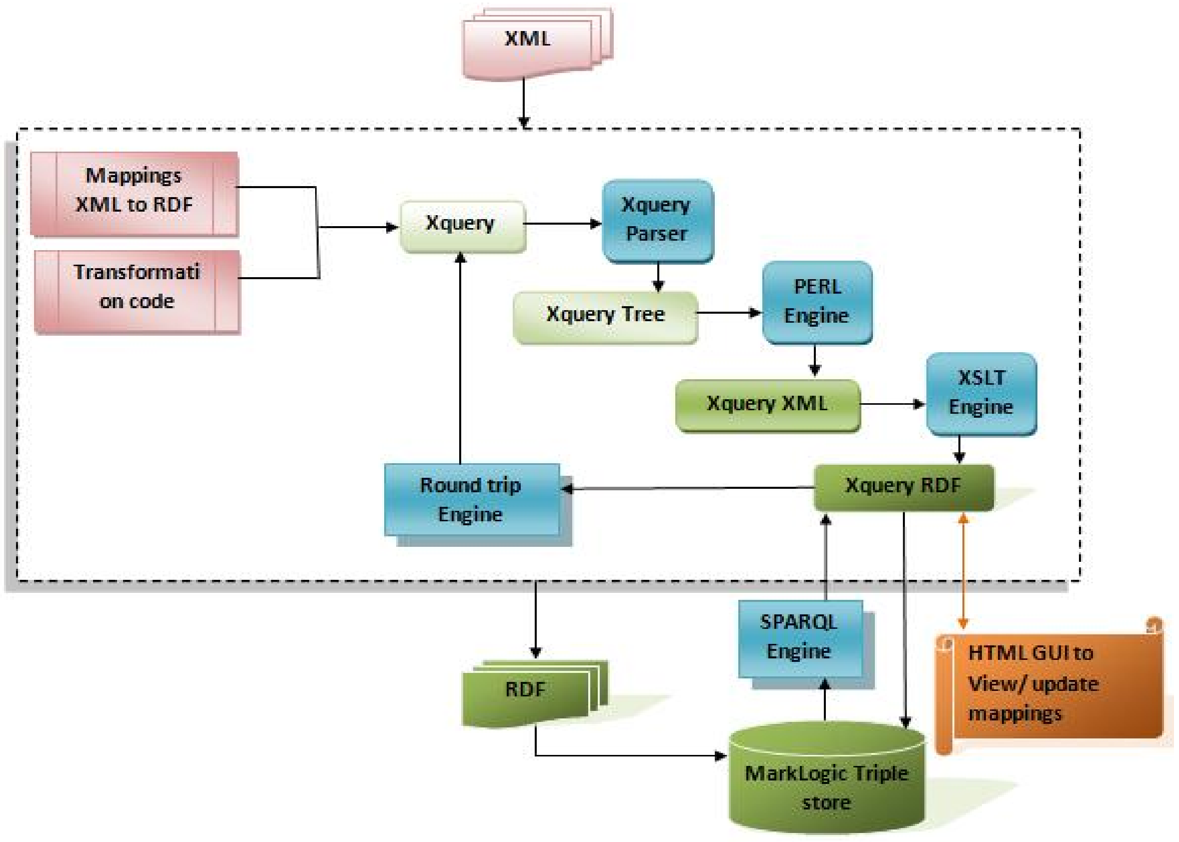


Figure 12: Technical architecture of proposed system

4.3.1 Component description

This section presents the details about the each technical component specified in **Figure 12.**

- XQuery: It is used to transform the XML to RDF. It contains all the mappings, the declaration of the mappings needs to be done in a specific format such that the mappings can be recognized in the “RDF structure of XQuery”. Details about the format are described in the implementation chapter.

- XQuery Parser: In this routine, the base XQuery will be parsed through the “XQuery parser” [33] which leads to the generation of XML tree structure (it is not an XML structure but the tree structure which has successive branching or subdivisions). This parser is created using the “XML representation of grammar” [50].

- Perl Engine: This engine performs PERL operations accompanied with XSLT operations. The tree structure of the XQuery generated by the “XQuery parser” gets transformed into an intermediate XML with the help of this engine. In this routine the PERL code is marking (not creating) the nesting for converting the tree to a well formed XML by reading the “tabs” and “enters” in the tree structure. Once the marking has been done by the PERL routine, the XSLT creates a well formed XML. The XSLT works on the basis of hierarchy levels marked by PERL, it is iterating over each element and checking its hierarchy value, if the hierarchical value of the current element is less than the previous element, the XSLT will make the current element, the child of the previous element.

- XSLT engine: This engine performs only the XSLT transformation. In this routine, the intermediate XML generated by the PERL engine get transformed into the desired RDF structure of the XQuery. The transformation between intermediate XML and “XQuery RDF” takes place based on the rules specified in the appendix A. This RDF is having the patterns through which the mappings (based on which XQuery is transforming the one format to another) can be identified and extracted for analysis and manipulation.

- Round Trip Engine: This engine performs XSLT operations on the “RDF structure of XQuery” to transform it back into the “XQuery expressions”. Whenever the “RDF structure of XQuery” gets manipulated by the user, this engine is responsible for reflecting those changes in the “XQuery expression” such that the data can be uplifted according to the changes. The transformation of “XQuery RDF” to “XQuery statements” takes place with the help of rules specified in appendix B.

- SPARQL engine: It takes the SPARQL query as an input from the user and executes it on the triple store. A provision has been provided in the interface through which user can query the triples. The execution of the SPARQL queries has been done with the help of the MarkLogic semantic library. Two modes of queries can be triggered 1) Read mode: user will not be able to update anything in the triple store. 2) Update mode: user cannot read anything if it is triggering the query in update mode, it is for updating the triples. Base URI: http://data.test.tcd.ie/resource of the graph should be used in order to analyze the triple of uplifted input. Base URI: http://xquery/XQrdf of the graph should be used in order to analyze or manipulate the triple of uplifted input.

4.3.2 Flow of operations

- **Step 1**: An input XML data set is required which needs to be transformed to RDF.
- **Step 2**: Establish the initial mappings of XML to RDF in the XQuery, which will be responsible for the transformation purpose.
- **Step 2.1**: This step will be carried out in parallel to step 2 as this involves following operations on XQuery to transform it RDF:
    - **Operation 1**: This involves parsing of the XQuery with the XQuery parser to transform it to tree structure.
    - **Operation 2**: This involves the transformation of tree structure to intermediate XML using PERL and XSLT.
    - **Operation 3**: This involves the transformation of intermediate XML to RDF using XSLT.

- **Step 3**: Transformation can be carried out based on the initially established mappings.

- **Step 4**: If any changes are made to the "XQuery RDF" either with the GUI or with the help of SPARQL, then the updated "XQuery RDF" needs to be transformed to the "XQuery expression" with the help of a round-trip engine.

4.4 Design Challenges and Resolutions

There are several challenges faced to achieve the research objectives. These challenges are discussed in this section. This section also describes the measures we took in order to overcome those challenges.

4.4.1 How to transform XQuery in an RDF data model automatically?

Challenge:

This is the primary challenge faced during the design. State of the art does not suggest any specific solution for transforming the XQuery to RDF. As a result, the combination of different technologies and tools has been used to achieve the research objectives.

Resolution:

No specific solution has been suggested in the State of art to transform XQuery to RDF. A combination of tools and techniques has been used in order to overcome this challenge. This is a crucial scenario in this approach as it is the main motivation behind the automatic solution for the re-composition and analysis of mappings. The transformation has been carried out with the help of following steps:

- **Step 1**: This involves parsing of the XQuery with the XQuery parser to transform it to tree structure. We are using the "XQuery parser" [33] for parsing the XQuery instead of "xq2xml tool". As [31] suggests that the output of xq2xml tool is not appropriate, if we need to transform it to another format. It has been noted that xq2xml tools removes some information of XQuery from the final output, which could be useful to the creator of XQuery.

- **Step 2**: This involves the transformation of tree structure to intermediate XML using PERL and XSLT. The detail of this is presented in the section 4.4.2.

- **Step 3**: Intermediate XML contains the elements and attributes based on the W3C recommended vocabulary specifications [50]. In order to transform this XML to RDF, the mappings need to be established. This mapping has been created in the development phase of the system and does not need any changes unless the bug is found. An XSLT has been used to for carrying out the transformation of intermediate XML to RDF, based on the rules specified in appendix A.

4.4.2 How to transform "tree structure of XQuery" to XML?

Challenge:

The Tree is the hierarchical structure, which has successive sub-division. This challenge is basically the part of the above challenge, but it is important to outline this as a different challenge as the solution to this challenge will be helpful in picking up the correct technology for a task of particular domains.

Resolution:

The output of the "XQuery parser" is the XML tree structure. In other words, it is a text file, but contains the hierarchy with the help of 'tabs' and 'enters'. PERL has been used in this system in order to convert the tree structure to intermediate XML. In this routine the PERL code is marking (not creating) the nesting for converting the tree to a well formed XML by reading the "tabs" and "enters" in the tree structure. Treating it with PERL's regular expressions has the following benefits:

- Complex regular expression can be easily handled with the help of PERL.

- Execution of regular expressions does not need any “tree structure of XML” to be parsed by an XML parser. As a result of which the huge amount of data can be easily processed.

- XML output can be easily delivered with the help of PERL.

4.4.3 XML should be uplifted to which RDF serialization format?

Challenge:
Chapter 2 describes the various serialization format of RDF. Most widely used formats of RDF are XML and Turtle. This research involves two types of data which need to be transformed to the RDF, 1) XQuery and 2) XML. So a serialization format needs to be decided such that mappings/ XQuery can be analyzed, manipulated and recomposed automatically.

Resolution:
In this research two types of data need to be transformed to RDF, 1) Input XML to RDF and 2) XQuery to RDF.

The mappings of Input XML to RDF are embedded into the XQuery. One of the objectives of this research is to analyze, manipulate and recompose the mappings. By transforming the XQuery to Turtle format will allow us to analyze and manipulate the mappings, but after manipulation it will not be convenient to recompose mappings/ XQuery from Turtle format automatically. So due to the re-composition issue, it has been decided to transform the XQuery to RDF/ XML format. As one of the biggest advantages of the XML is the flexibility it provides for structuring the information. Keeping the XQuery in RDF / XML not only helped in easy re-composition of XQuery, but also we managed to extract mapping information out of it for rendering purpose.

As we are using the XQuery for the transformation purpose, it has been decided to transform the Input XML to RDF/ XML format. From the design perspective, it is also a good practice to keep the consistency within the system.

4.4.4 How to treat blank nodes in the uplifted XML?

Challenge:

A blank node in RDF is a node which does not contain any data. It is important to handle the blank nodes in RDF. It is harder to manipulate RDF containing a blank node. As a result, consuming the RDF containing a blank node can create problems for the data consumers.

Resolution:

Handling of blank node can be complicated while consuming RDF data. It is also not possible for us to decide which blank node needs to be in RDF or not. It's the owner of the data which provides this information. So for handling this challenge we have provided a mechanism in the system to handle the blank node. The owner of data or user of the system can add the node name in the configuration whose blank node needs to be preserved. By default the system is configured to delete all the blank nodes. The configuration file can be found in the attached CD (\lifting\assets\code\Blanknodes.xml).

4.5 Summary

This chapter discussed the requirements for the proposed system. Technical architecture has also been discussed in this chapter for the better understanding of each and every technical component of the system. The major challenges faced during the design phase and the measures to overcome those challenges have also been listed in this chapter.

Chapter 5

Implementation

In this chapter, we describe the implementation of the design of the proposed system. This implementation is completely based on the technical architecture specified in section 4.3. This chapter begins with the introduction of the tools and technologies used in the implementation phase.

5.1 Tools and Technologies

We are using various tools and technologies in order to achieve all the objectives of this study. These technologies comprise of XML technologies, semantic technologies and Query languages. In particular, the following are the tools/ technologies used in this implementation.

- **XML**: It is a text format derived from the SGML. The purpose of XML is to store and transport the data [42]. XML provides the flexibility to structure the information effectively. XML is being used in this implementation extensively. The proposed system is compatible with the input XML dataset. The final output will be in XML serialization format of RDF.

- **RDF**: It is the main building block of the Semantic Web. It describes the resources with the help of statements, these statements are in the form of Subject-Predicate-Object expression. These expressions are known as triples. In the triples subject is the resource about which the triple is describing, predicate defines the property of the resource, and object is the value of the property which could be a literal value or it can redirect to the subject of another resource [5]. In this study the input XML dataset and XQuery code are uplifted to RDF.

- **XQuery**: It is a language which is derived from an XML query language called Quilt. It inherits the features of various languages such as XPATH [45], XQL [44], SQL [46] and ODMG [47] [43].

- **MarkLogic**: It is a suite of database, Web server and search engine. It has the capability to store and manages the documents like XML, JSON and Semantic data (RDF triples). In this implementation, we are using MarkLogic to deploy the application, storing the triples and managing the RDF/ XML documents [41].

- **XSLT**: It is a transformation language which is used to transform one XML format to another [48]. In this study XSLT is being used in several intermediate transformation scenarios. One of the major uses of XSLT in this implementation is to transform the "RDF structure of XQuery" back into the "XQuery expressions".

- **PERL**: It is a programming language especially designed for text processing. In this implementation, we are using PERL for transforming the "tree structure of XQuery" to an XML structure.

- **SPARQL**: It is a query language for the RDF graphs, SPARQL query needs a triple pattern which is known as basic graph pattern [8]**.** SPARQL query contains two parts, first is "SELECT" clause which is used to construct the result, second part is "WHERE" clause used the triple pattern provided to match against the RDF provided. In this implementation, we are providing a mechanism for the user to analyze and manipulate the triples of transformed RDF.

5.2 Conventions for implementation

The proposed system has been implemented using MarkLogic [41]. MarkLogic is a NoSQL database and the language which is used to query it, is XQuery. This implementation is based on the technical architecture specified in section 4.3.

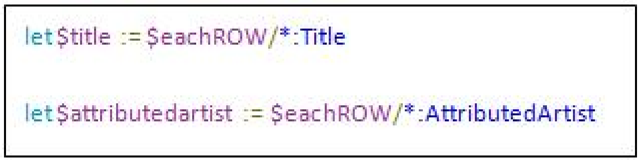
This implementation relies on the following conventions:

- **Input**: it should always be in the XML format. Lifting of any other input format has not been accommodated in this implementation.

- **XQuery**: The XQuery used for transforming the input XML to RDF should follow the below conventions in order to define mappings. This is important as defining mappings according to these conventions will help in extracting the mapping information from the "RDF structure of XQuery".

  - The variable name should be "$eachROW", which contains the XML structure of the resource, employee, citation, etc. depends on the input dataset which is in use.

  - For selecting an element from input XML for mapping, the use of "$eachROW" variable in XPath is mandatory. Please refer the **Figure 13** for instances. This is recommended because in "RDF of XQuery" it can be identified, what all input XML elements have been mapped.

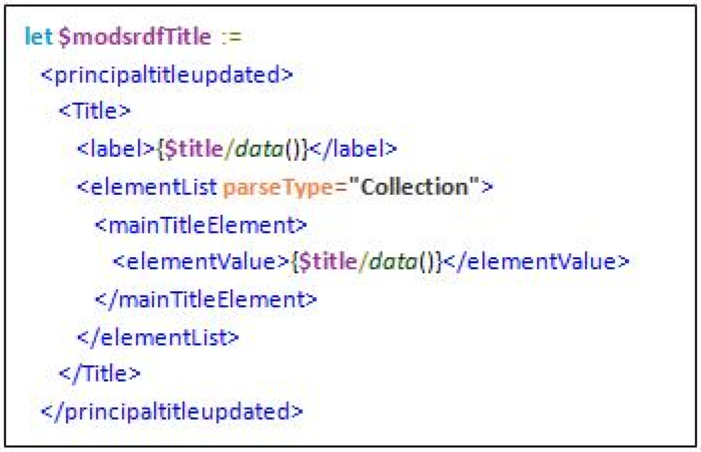


```
let $title := $eachROW/*:Title

let $attributedartist := $eachROW/*:AttributedArtist
```

Figure 13: Selection of "Title" and "AttributedArtist" from the input XML for mapping

  - For defining the corresponding RDF for the selected input XML elements, the use of the variable name which contains "rdf" is mandatory. Please refer the **Figure 14** for instances. This is recommended because in "RDF of XQuery" it can be identified, what is the corresponding RDF structure of all input XML elements.



```
let $modsrdfTitle :=
  <principaltitleupdated>
    <Title>
      <label>{$title/data()}</label>
      <elementList parseType="Collection">
        <mainTitleElement>
          <elementValue>{$title/data()}</elementValue>
        </mainTitleElement>
      </elementList>
    </Title>
  </principaltitleupdated>
```

Figure 14: Corresponding RDF structure for “Title” element of input XML

- The complete XQuery which transform the FileMaker XML to MODS RDF can be found in the attached CD (\lifting\assets\code\fileMakerToModsRDF.xquery).

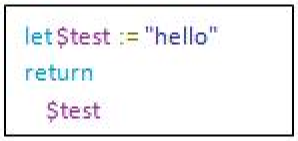
5.3 Use cases implementation description

The major parts in this implementation are 1) Transformation of XQuery to RDF 2) Transformation of Input XML to RDF 3) Rendering mappings in GUI 4) Analysis and manipulation of triples 5) if we need to uplift a new dataset to RDF. A mechanism to switch dataset has been provided in this system. This section discusses the implementation of these processes in details.

5.3.1 Use case 1: XQuery to RDF

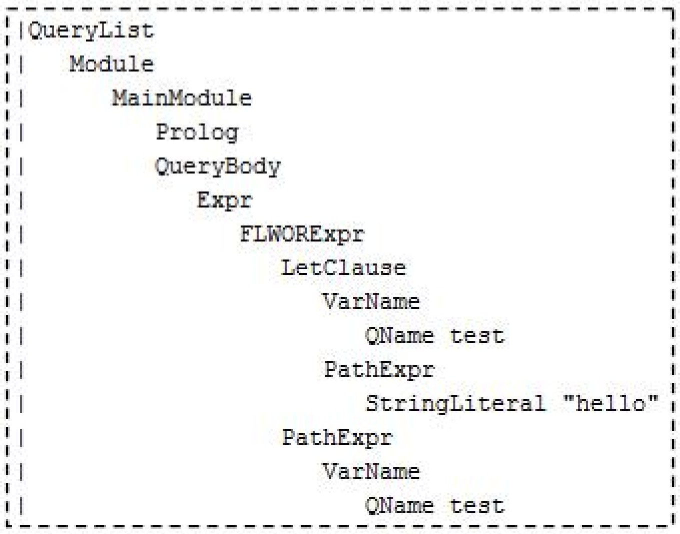
This is the most crucial part of this implementation, as the most important objective of this research is to represent the mappings in RDF structure. Representing the mappings in RDF enables the user to analyze and manipulate them automatically through SPARQL queries. The transformation of XQuery to RDF has been carried out in following steps

- **Step 1**: After finalizing the mappings, the base XQuery needs to be established. Once it is in place, it will be parsed through the "XQuery parser" [33] which leads to the generation of XML tree structure (it is not an XML structure but the tree structure which has successive branching or subdivisions). The generated tree is based on the vocabulary specification [50] recommended by W3C. The parser can be run with the help of following commands in the terminal.
    - **< java -jar xquery.jar >** it will prompt for an expression.
    - **< java -jar xquery.jar -file [filename]>** it will parse an XQuery file and tree will be generated on the terminal itself.
    - **Figure 15** and **Figure 16**, shows the transformation of sample XQuery to its tree structure.



```
let $test := "hello"
return
  $test
```

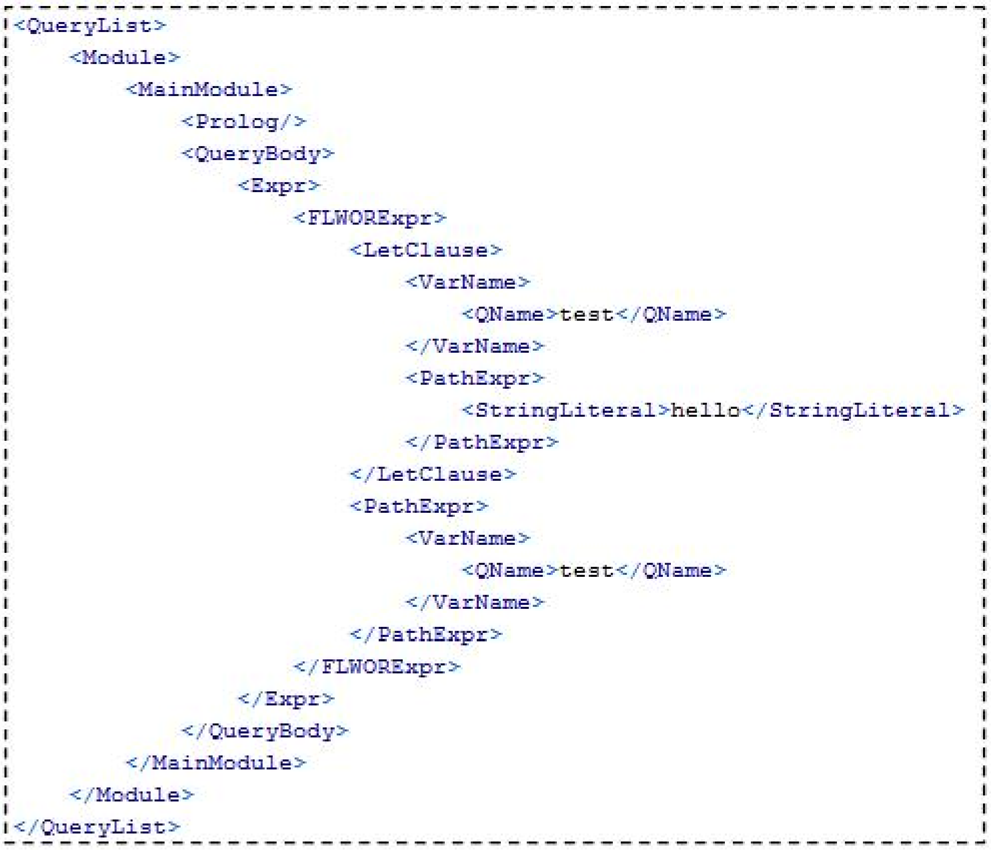
Figure 15: A sample XQuery for instance



```
|QueryList
|   Module
|      MainModule
|         Prolog
|         QueryBody
|            Expr
|               FLWORExpr
|                  LetClause
|                     VarName
|                        QName test
|                     PathExpr
|                        StringLiteral "hello"
|                  PathExpr
|                     VarName
|                        QName test
```

Figure 16: Tree structure of sample XQuery

- **Step 2**: In this step, the PERL operations accompanied with XSLT operations has been applied to the tree structure generated in the previous step. With every line break a new element starts in the generated tree. The PERL routine in this engine is iterating over each line break and marking the hierarchy of the element by counting the number of “tabs” before its name. The complete PERL code can be found in the attached CD (\lifting\assets\code\XqueryToXML.pl). Once the marking of hierarchy for each element has been done, an XSLT is executed, which transform the output generated by the PERL routine. The XSLT is creating the nested XML structure based on the hierarchy level marked by the PERL routine. The complete XSLT code can be found in the attached CD (\lifting\assets\code\treetoxml.xsl). **Figure 17** shows the output generated by the XSLT transformation.

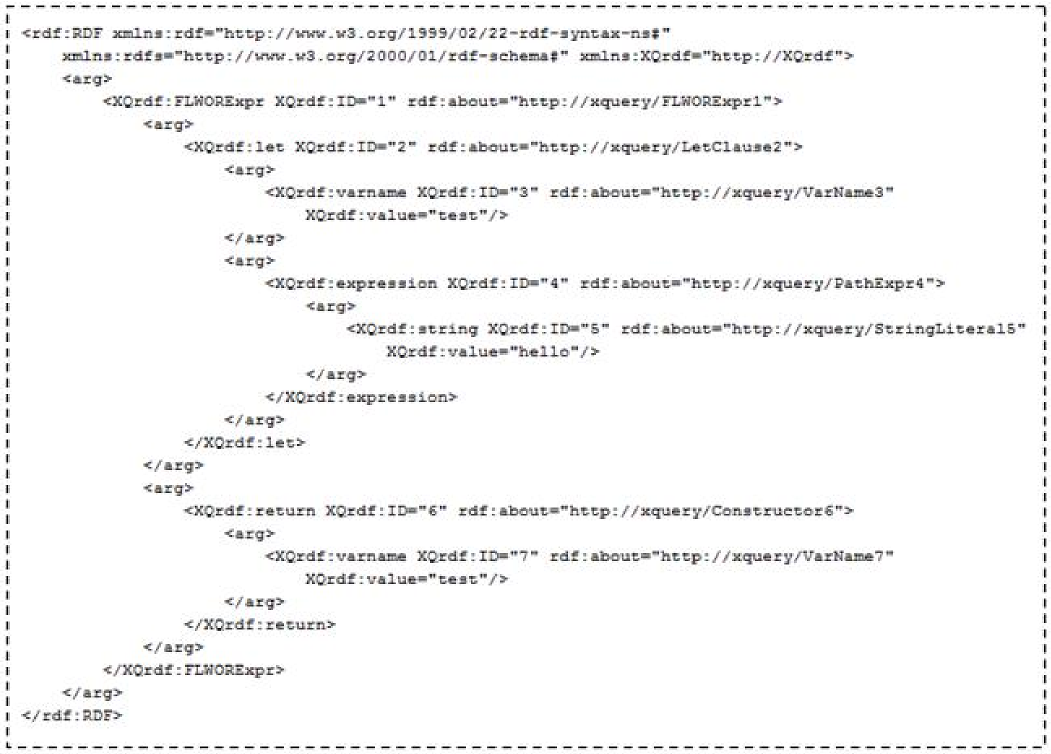


```
<QueryList>
    <Module>
        <MainModule>
            <Prolog/>
            <QueryBody>
                <Expr>
                    <FLWORExpr>
                        <LetClause>
                            <VarName>
                                <QName>test</QName>
                            </VarName>
                            <PathExpr>
                                <StringLiteral>hello</StringLiteral>
                            </PathExpr>
                        </LetClause>
                        <PathExpr>
                            <VarName>
                                <QName>test</QName>
                            </VarName>
                        </PathExpr>
                    </FLWORExpr>
                </Expr>
            </QueryBody>
        </MainModule>
    </Module>
</QueryList>
```

Figure 17: Intermediate XML structure of sample XQuery

- **Step 3**: In this step, the intermediate XML generated in the previous step get transformed into the desired RDF structure of the XQuery. This transformation is carried out with the help of XSLT, based on the rules specified in appendix A. The complete XSLT code used for this transformation can be found in the attached CD (\lifting\assets\code\XQueryXMLToRDFNew.xsl). We are using SAXON 9 for this transformation; it can be invoked with the help of following terminal command.
    - **< java -jar jar/saxon9pe.jar {input XML path} {XSLT path } {output RDF path}>**

  Figure 18 shows the RDF generated after this step.



```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#"
    xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" xmlns:XQrdf="http://XQrdf">
    <arg>
        <XQrdf:FLWORExpr XQrdf:ID="1" rdf:about="http://xquery/FLWORExpr1">
            <arg>
                <XQrdf:let XQrdf:ID="2" rdf:about="http://xquery/LetClause2">
                    <arg>
                        <XQrdf:varname XQrdf:ID="3" rdf:about="http://xquery/VarName3"
                            XQrdf:value="test"/>
                    </arg>
                    <arg>
                        <XQrdf:expression XQrdf:ID="4" rdf:about="http://xquery/PathExpr4">
                            <arg>
                                <XQrdf:string XQrdf:ID="5" rdf:about="http://xquery/StringLiteral5"
                                    XQrdf:value="hello"/>
                            </arg>
                        </XQrdf:expression>
                    </arg>
                </XQrdf:let>
            </arg>
            <arg>
                <XQrdf:return XQrdf:ID="6" rdf:about="http://xquery/Constructor6">
                    <arg>
                        <XQrdf:varname XQrdf:ID="7" rdf:about="http://xquery/VarName7"
                            XQrdf:value="test"/>
                    </arg>
                </XQrdf:return>
            </arg>
        </XQrdf:FLWORExpr>
    </arg>
</rdf:RDF>
```

Figure 18: RDF structure of sample XQuery

As we are creating it in a completely automated way, the commands to execute all the above steps have been stated in a shell script. To perform the transformation of XQuery to RDF, the shell script needs to be executed. The script can be found in the attached CD (\lifting\assets\Processing.bat)

5.3.2 Use case 2: Input XML to RDF

To uplift the information automatically, is the main objective of this research. We are using the XQuery in this implementation to uplift the data into RDF. To transform the input XML to RDF, a base XQuery needs to be in place. This XQuery contains the mappings of input XML to RDF. Mapping rules have to be finalized before establishing the base XQuery. In this implementation, we are using “SAXON 9” in order to transform XML to RDF using XQuery. The XQuery used for the transformation of FileMaker to MODS RDF can be found in the CD (\lifting\assets\code\fileMakerToModsRDF.xquery). To invoke the transformation, the command to execute XQuery has been stated in the shell script. The execution of this shell script has been bound to a button in the MarkLogic based application. The MarkLogic code can be referred on the “\lifting\assets\HttpServer\code\transformation.xqy” in the attached CD.

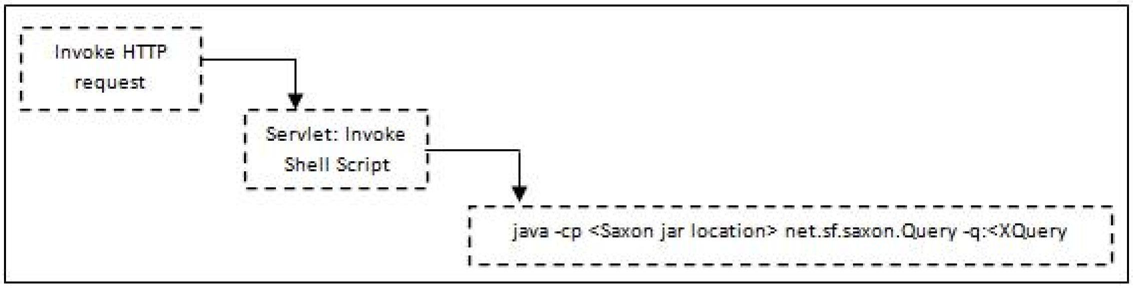
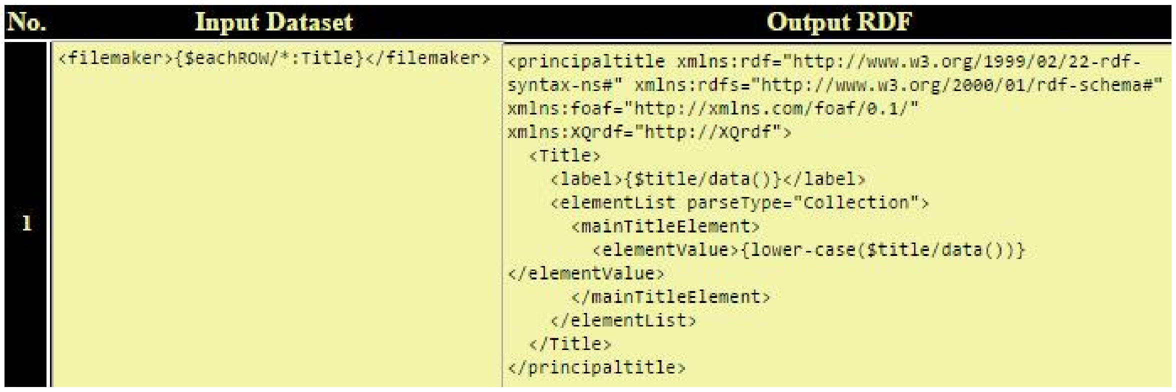

Invoke HTTP request
Servlet: Invoke Shell Script
java -cp <Saxon jar location> net.sf.saxon.Query -q:<XQuery

Figure 19: Execution of transformation of input XML to RDF

5.3.3 Use case 3: Rendering mappings in GUI

One of the objectives of this research is to manipulate the mappings. In this project we provide three ways to manipulate the mappings 1) GUI whose implementation is described in this section, 2) By updating the triples of “XQuery RDF” using SPARQL queries, implementation of this is described in the section 5.3.4, 3) By making changes in the base XQuery. Implementation of how to switch or update the XQuery has been discussed in the section 5.3.5.

We have provided an interface where users can view and update mappings manually. This interface is designed in HTML which the help of MarkLogic code. The interface contains a table in which the first column is “No.” which contains the count of mapping, second column is “Input Dataset” which contains the XPATH of input element which needs to be mapped and the third column is “Output RDF” which contains the output RDF structure for the corresponding input element. Please refer **Figure 20**.



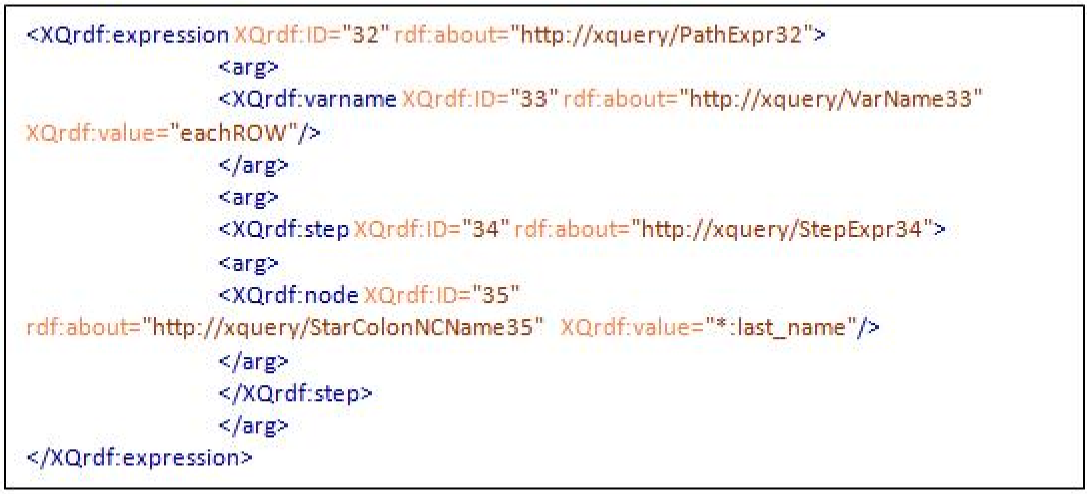
No.	Input Dataset	Output RDF
1	`<filemaker>{$eachROW/*:Title}</filemaker>`	`<principaltitle xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#" xmlns:rdfs="http://www.w3.org/2000/01/rdf-schema#" xmlns:foaf="http://xmlns.com/foaf/0.1/" xmlns:XQrdf="http://XQrdf"> <Title> <label>{$title/data()}</label> <elementList parseType="Collection"> <mainTitleElement> <elementValue>{lower-case($title/data())} </elementValue> </mainTitleElement> </elementList> </Title> </principaltitle>`

Figure 20: Interface to view/ update mappings manually

The mapping information is extracted from the “XQuery RDF”. As the base XQuery needs to be established based on the conventions specified in the section 5.2. These conventions enable the code to identify the corresponding input elements and output RDF structure. The MarkLogic code identifies the mappings and transforms these into the readable format while rendering it to HTML as shown in **Figure 20**. The complete code for this use case can be found in the attached CD (\lifting\assets\HttpServer\code\viewmappings.xqy)

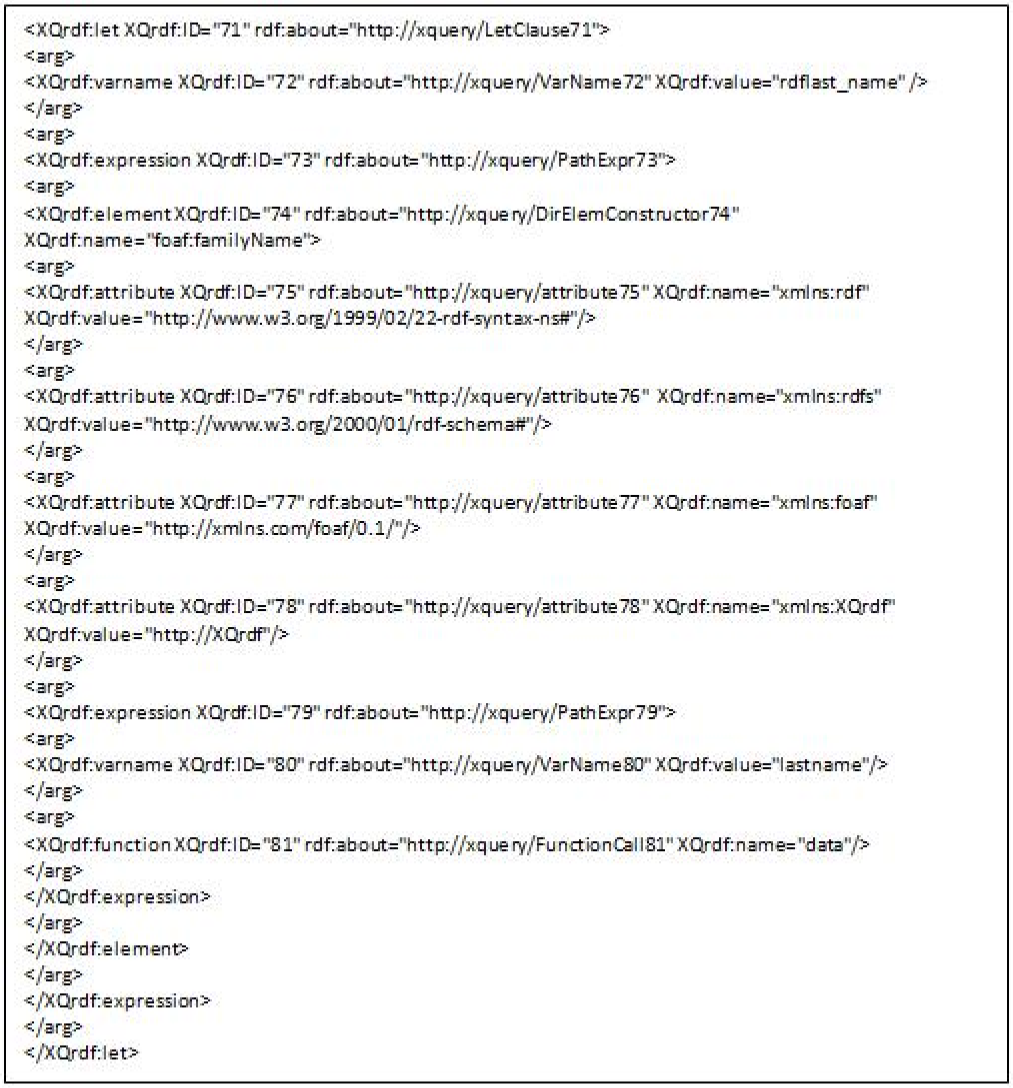
Attribute {XQrdf:value="eachROW"} enables the code to understand that an input element has been specified here for mapping. Complete details about the input element can be extracted from the other child elements of <XQrdf:expression>. Please refer **Figure 21** for the complete “RDF structure of input element <Last_name>”.

The corresponding “RDF structure of the output” of input element can be identified by the variable whose name contains “rdf” (instance: XQrdf:value="rdflast_name"). Another join to this condition is that the input element has to be called in the <expression> element of “RDF structure of output” (instance: <XQrdf:varname XQrdf:ID="" rdf:about="" XQrdf:value="lastname"/>). Please refer **Figure 22** for the complete “RDF structure of the output” of <Last_name>.



```
<XQrdf:expression XQrdf:ID="32" rdf:about="http://xquery/PathExpr32">
                    <arg>
                    <XQrdf:varname XQrdf:ID="33" rdf:about="http://xquery/VarName33"
XQrdf:value="eachROW"/>
                    </arg>
                    <arg>
                    <XQrdf:step XQrdf:ID="34" rdf:about="http://xquery/StepExpr34">
                    <arg>
                    <XQrdf:node XQrdf:ID="35"
rdf:about="http://xquery/StarColonNCName35"   XQrdf:value="*:last_name"/>
                    </arg>
                    </XQrdf:step>
                    </arg>
</XQrdf:expression>
```

Figure 21: Input element “last_name” that has to be mapped



```xml
<XQrdf:let XQrdf:ID="71" rdf:about="http://xquery/LetClause71">
<arg>
<XQrdf:varname XQrdf:ID="72" rdf:about="http://xquery/VarName72" XQrdf:value="rdflast_name" />
</arg>
<arg>
<XQrdf:expression XQrdf:ID="73" rdf:about="http://xquery/PathExpr73">
<arg>
<XQrdf:element XQrdf:ID="74" rdf:about="http://xquery/DirElemConstructor74"
XQrdf:name="foaf:familyName">
<arg>
<XQrdf:attribute XQrdf:ID="75" rdf:about="http://xquery/attribute75" XQrdf:name="xmlns:rdf"
XQrdf:value="http://www.w3.org/1999/02/22-rdf-syntax-ns#"/>
</arg>
<arg>
<XQrdf:attribute XQrdf:ID="76" rdf:about="http://xquery/attribute76" XQrdf:name="xmlns:rdfs"
XQrdf:value="http://www.w3.org/2000/01/rdf-schema#"/>
</arg>
<arg>
<XQrdf:attribute XQrdf:ID="77" rdf:about="http://xquery/attribute77" XQrdf:name="xmlns:foaf"
XQrdf:value="http://xmlns.com/foaf/0.1/"/>
</arg>
<arg>
<XQrdf:attribute XQrdf:ID="78" rdf:about="http://xquery/attribute78" XQrdf:name="xmlns:XQrdf"
XQrdf:value="http://XQrdf"/>
</arg>
<arg>
<XQrdf:expression XQrdf:ID="79" rdf:about="http://xquery/PathExpr79">
<arg>
<XQrdf:varname XQrdf:ID="80" rdf:about="http://xquery/VarName80" XQrdf:value="lastname"/>
</arg>
<arg>
<XQrdf:function XQrdf:ID="81" rdf:about="http://xquery/FunctionCall81" XQrdf:name="data"/>
</arg>
</XQrdf:expression>
</arg>
</XQrdf:element>
</arg>
</XQrdf:expression>
</arg>
</XQrdf:let>
```

Figure 22: mapping rule (RDF structure) for Input element "last_name"

As soon as the user submits the changes in mappings, the updated mappings will be sent to MarkLogic through form. Then MarkLogic will make changes in the existing "XQuery RDF" according to the updated mapping information.

5.3.4 Use case 4: Analysis and manipulation of triples

The above use cases is for the uplifting of "XML and XQuery" to RDF. In this use case triples of the uplifted XML will be stored in the MarkLogic triple store. A provision for the user has been provided to query these triples through SPARQL. To query the graph related to “RDF of XQuery”, URI: <http://xquery/XQrdf> is to be used. To query the graph related to uplifted XML, URI: <http://data.test.tcd.ie/resource> is to be used.

An interface has been provided to the user through which user can write and invoke the SPARQL query, please refer **Figure 23**.

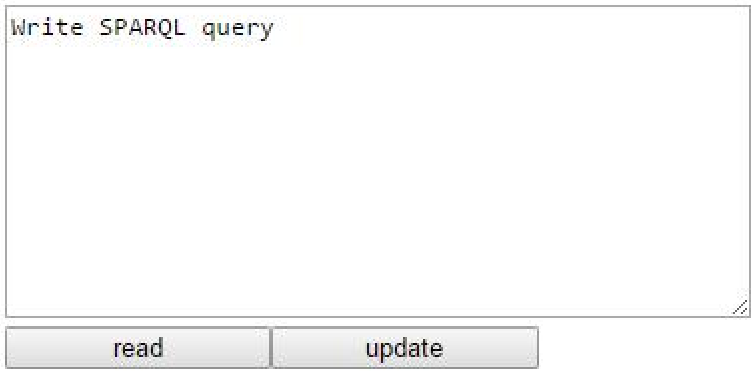


Figure 23: Interface to execute SPARQL queries

Semantic library of MarkLogic has been used in order to implement this functionality. It allows both read and update type of SPARQL queries to be executed. The SPARQL query written by the user goes to the MarkLogic end with the help of forms. MarkLogic retrieves the query and sends the request to execute query using its semantic library. Once the query has been processed, the response to the query will be sent back to render on the interface. Please refer **Figure 24** to understand the execution of a SPARQL query.

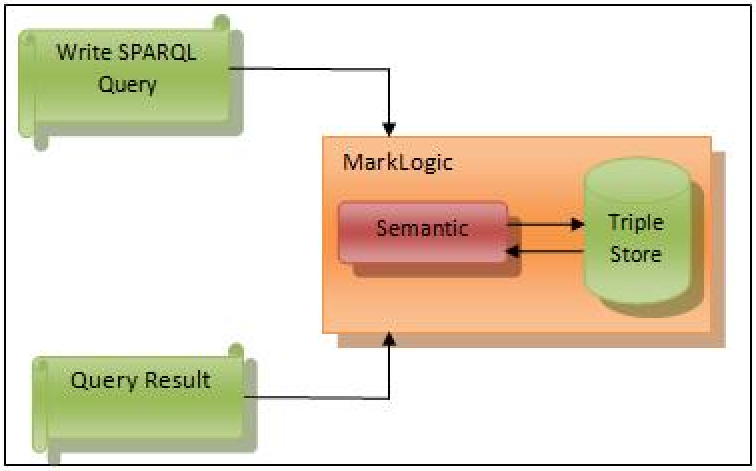
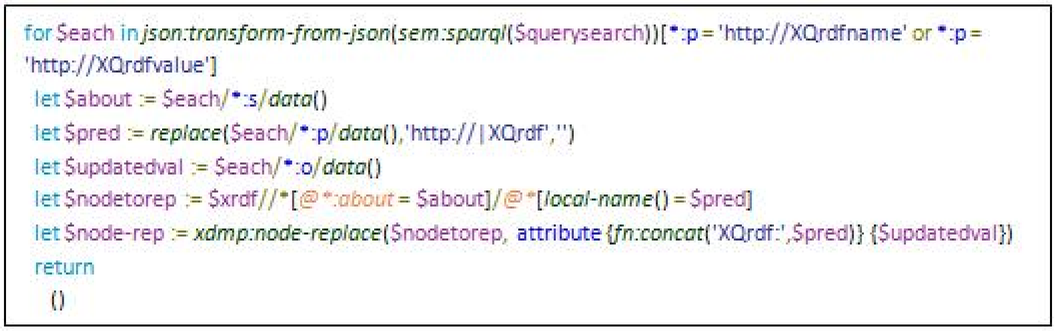


Figure 24: Execution of SPARQL queries

SPARQL read query can be used to analyze the XQuery or the uplifted XML. A lot of useful information about the transformation code can be extracted by analyzing these triples. In order to manipulate these triple one has to invoke the SPARQL update query. The changes made in the “XQuery triples” will be reflected in the “XQuery RDF” with the help of code mentioned in the **Figure 25**. Once the “XQuery RDF” is updated with the changes done in triples, the XQuery will be recomposed using the round trip engine to uplift the data according to the updated mapping rules. The complete code of the round trip engine can be found in attached CD (\lifting\assets\code\RDFTToXQueryNew.xsl)



```
for $each in json:transform-from-json(sem:sparql($querysearch))[*:p = 'http://XQrdfname' or *:p =
'http://XQrdfvalue']
 let $about := $each/*:s/data()
 let $pred := replace($each/*:p/data(),'http://|XQrdf','')
 let $updatedval := $each/*:o/data()
 let $nodetorep := $xrdf//*[@*:about = $about]/@*[local-name() = $pred]
 let $node-rep := xdmp:node-replace($nodetorep, attribute {fn:concat('XQrdf:',$pred)} {$updatedval})
 return
   ()
```

Figure 25: code snippet to update "XQuery RDF" based on triple’s changes

5.3.5 Use case 5: Switching to new dataset

This is another way of updating mapping. By this way the system can be configured to uplift a different dataset as well. For updating the mapping in XQuery, the changes in the XQuery have to be made according to the conventions specified in section 5.2. Once the changes are done, this XQuery needs to replace the existing base XQuery in the system. For this update we have provided a mechanism in the system through which user can upload the updated XQuery or completely different XQuery. Please refer the **Figure 26**.

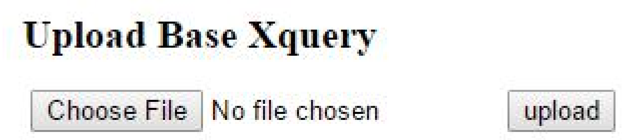
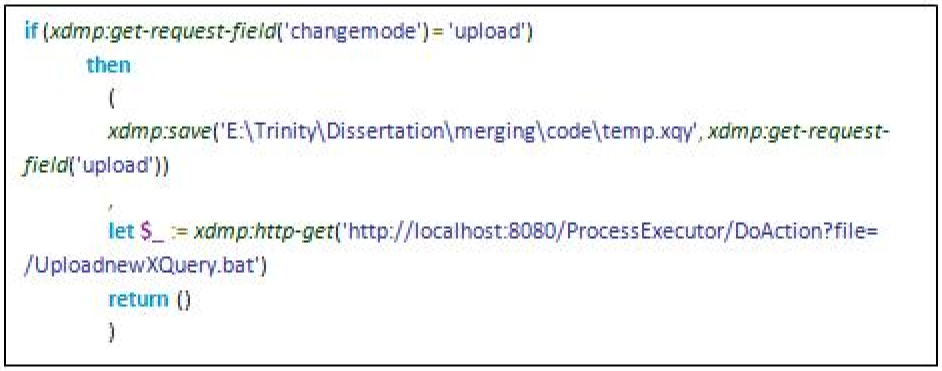


Figure 26: Interface to upload updated XQuery

The implementation of this upload is done through “input type file” element of HTML. The updated XQuery will be sent to MarkLogic using forms. MarkLogic retrieves the updated XQuery and replace the existing XQuery in the system with the updated one. Please refer the code in the **Figure 27.**



```
if (xdmp:get-request-field('changemode') = 'upload')
        then
          (
          xdmp:save('E:\Trinity\Dissertation\merging\code\temp.xqy', xdmp:get-request-
field('upload'))
          ,
          let $_ := xdmp:http-get('http://localhost:8080/ProcessExecutor/DoAction?file=
/UploadnewXQuery.bat')
          return ()
          )
```

Figure 27: code to replace the existing XQuery with updated one

5.4 Summary

In this chapter, the details about the tools and technologies used in this implementation have been captured. It also specified the conventions used to create the base XQuery at the time of finalizing the mappings. This chapter describes the implementation of the technical architecture presented in the section 4.3. The implementation of the following use cases has been discussed in this chapter:

- **XQuery to RDF**: It is multiphase process, in the first phase, the XQuery gets parsed by the XQuery parser. Few transformation operations have been applied to the output of the XQuery parser to convert it to RDF.

- **Input XML to RDF**: The implementation of this has been carried out with the help of base XQuery created at the time of finalizing the mappings.

- **Rendering mapping in GUI**: The implementation of this involves the extraction of mapping information from the “XQuery RDF” and rendering it on HTML. It also allows the user to manipulate this information.

- **Analysis and manipulation of triples**: The implementation of this use case involves, querying the triple store based on the user defined queries. The execution of the SPARQL queries takes place with the help of semantic library of the MarkLogic. The implementation takes care of both read and update SPARQL queries.

- **Switching to new dataset**: In this implementation, the updated base XQuery or new base XQuery is able to replace the existing base XQuery in the system. It provides a mechanism to upload the different XQuery in the system and able to see the changes in the mappings through the GUI.

Chapter 6

Evaluation

This chapter describes the evaluation of the developed prototype of the proposed system. The first section of this chapter describes the various test cases we have run on this system in order to check if the system meets the key requirements or not. The second section of this chapter evaluates the system from the user's perspective. It involves evaluation of the system based on the 1) Ease of task completion, 2) Time to complete the task, and 3) Support information provided to carry out the task.

The tests described in this section were executed on a Lenovo system which has the following configuration:

- Processor: Intel i3, 1.80 GHz
- RAM: 4.00 GB
- System type: Windows 10, 64-bit

The developed prototype has been deployed on the MarkLogic Server 8.0-5.2. The intermediate transformations in the system have been carried out with the help of Strawberry PERL (v5.24.0) and XSLT 2.0 (SAXON 9).

6.1 Evaluation: Test cases

For executing these test cases, the Filemaker XML has been selected as an input dataset. We have run all the test cases described in the subsections to check the system is capable enough to achieve all the objectives mentioned in the section 1.2. For carrying out these test cases following condition needs to be satisfied:

- FileMaker XML should be present in the directory "../Filemaker".
- Base XQuery should be present in the directory "../code" in order to perform the transformation.
- Saxon 9 jar should be present in the directory "../jar".
- The XQuery parser should be present in the parent directory of the application root directory.
- Supporting XSLTs/ PERL code should be present in the directory "../code", which are used in the intermediate transformation.
- Tomcat apache needs to be running with the process executor code (WAR responsible for invoking the shell script.)

***Note:** All paths used in this section are relative to the application root directory.*

6.1.1 Test case 1: Uplifting input XML to RDF

Purpose: The purpose of this is to test the capability of the system to support the user in uplifting the data (XML) to the RDF (RDF/ XML) automatically based on the specified mappings in base XQuery.

Execution: The transformation should be executed after clicking the only transformation icon present on the "Transformation" page of the application.

Expected outcome: We expected that the System should be able to uplift the XML to RDF based on the mappings (in the case of FileMaker dataset: FileMaker to MODS RDF) specified in the base XQuery. The uplift process should be completed without throwing any exception. The user should be able to locate the transformed RDF in the directory "../modsRDF".

Actual outcome: The uplift process of XML has been completed successfully. The user was able to locate the uplifted XML in "../modsRDF".

6.1.2 Test case 2: Update mapping through user GUI-I

Purpose: The purpose of this is to test the capability of the system to support the user in changing the mapping rules. The user is changing the mapping rules through GUI and performing the transformation based on the updated mappings. In this particular test case the user is changing the element name in the mapping rule.

Execution: The user performed this test by changing the mapping rule specified for <Title> element. The changes performed are as follow**s:**

Intia rule:

<principaltitle>

<Title>...</Title>

</principaltitle>

Updated rule:

<principaltitle>

<TitleUpdated>...</TitleUpdated>

</principaltitle>

Expected outcome: We expected that the system should be successful in updating the mapping and uplift the XML according to the updated mapping rule. The user should be able to locate the changes in the transformed RDF. In this particular test case the user should be able to locate <TitleUpdated>...</TitleUpdated>.

Actual outcome: User was able to update the mapping successfully. The transformation performed after updating the mapping was completed without any exception. <TitleUpdated>...</TitleUpdated> has been located by the user in the transformed output.

6.1.3 Test case 3: Update mapping through user GUI-II

Purpose: The purpose of this is to test the capability of the system to support the user in changing the mapping rules. The user is changing the mapping rules through GUI and performing the transformation based on the updated mappings. In this particular test case the user is adding an attribute in the element of the mapping rule.

Execution: The user performed this test by changing the mapping rule specified for <Title> element. The changes performed are as follow**s:**

Intia rule:

```
<principaltitle>
 <TitleUpdated>… </TitleUpdated>
</principaltitle>
```

Updated rule:

```
<principaltitle>
 <TitleUpdated update="twice">...</TitleUpdated>
</principaltitle>
```

Expected outcome: We expected that the system should be successful in updating the mapping and uplift the XML according to the updated mapping rule. The user should be able to locate the changes in the transformed RDF. In this particular test case the user should be able to locate <TitleUpdated update="twice">...</TitleUpdated>.

Actual outcome: User was able to add the attribute in the mapping rule successfully. The transformation performed after updating the mapping was completed without any exception. <TitleUpdated update="twice">...</TitleUpdated> has been located by the user in the transformed output.

6.1.4 Test case 4: Update mapping through user GUI-III

Purpose: The purpose of this is to test the capability of the system to support the user in changing the mapping rules. The user is changing the mapping rules through GUI and performing the transformation based on the updated mappings. In this particular test case the user is manipulating the value of an element in the mapping rule.

Execution: The user performed this test by changing the mapping rule specified for <Title> element. The changes performed are as follows:

Intia rule:

```
<principaltitle>
 <TitleUpdated update="twice">
   <label>{$title/data()}</label>
   <elementList parseType="Collection">
     <mainTitleElement>
             <elementValue>{$title/data()}</elementValue>
     </mainTitleElement>
   </elementList>
 </TitleUpdated>
</principaltitle>
```

Updated rule:

```
<principaltitle>
 <TitleUpdated update="twice">
  <label>{$title/data()}</label>
  <elementList parseType="Collection">
   <mainTitleElement>
        <elementValue>{upper-case($title/data())}</elementValue>
   </mainTitleElement>
  </elementList>
 </TitleUpdated>
</principaltitle>
```

Expected outcome: We expected that the system should be successful in updating the mapping and uplift the XML according to the updated mapping rule. The user should be able to locate the changes in the transformed RDF. In this particular test case the user should be able to locate the value of <elementValue>...</elementValue> element in upper case.

Actual outcome: User was able to update the mapping rule successfully. The transformation performed after updating the mapping was completed without any exception. The value of <elementValue>...</elementValue> element has been located in upper case by the user in the transformed output.

6.1.5 Test case 5: Update mapping through user GUI-IV

Purpose: The purpose of this is to test the capability of the system to support the user in changing the mapping rules. The user is changing the mapping rules through GUI and performing the transformation based on the updated mappings. In this particular test case the user is adding a new element in already defined mapping rule.

Execution: The user performed this test by changing the mapping rule specified for <Title> element. The changes performed are as follows:

Intia rule:

```
<principaltitle>
 <TitleUpdated update="twice">
   <label>{$title/data()}</label>
   <elementList parseType="Collection">
     <mainTitleElement>
            <elementValue>{upper-case($title/data())}</elementValue>
     </mainTitleElement>
   </elementList>
 </TitleUpdated>
</principaltitle>
```

Updated rule:

```
<principaltitle>
 <TitleUpdated update="twice">
   <label>{$title/data()}</label>
   <elementList parseType="Collection">
     <mainTitleElement>
          <elementValue>{upper-case($title/data())}</elementValue>
          <smallCaseElementValue>{lower-case($title/data())}</smallCaseElementValue>
     </mainTitleElement>
   </elementList>
 </TitleUpdated>
</principaltitle>
```

Expected outcome: We expected that the system should be successful in updating the mapping and uplift the XML according to the updated mapping rule. The user should be able to locate the changes in the transformed RDF. In this particular test case the user should be able to locate the <smallCaseElementValue>...</smallCaseElementValue> element.

Actual outcome: User was able to update the mapping rule successfully. The transformation performed after updating the mapping was completed without any exception. The <smallCaseElementValue>...</smallCaseElementValue> element has been located by the user in the transformed output.

6.1.6 Test case 6: Changing mapping by updating triples-I

Purpose: The purpose of this is to test the capability of the system to support the user in changing the mapping rules. The user is changing the mapping rules by updating triples of “XQuery RDF” (base URI: http://xquery/XQrdf) using SPARQL update queries. The user is then performing the transformation based on the updated mappings. In this particular test case the user is changing an element name in already defined mapping rule.

Execution: The user performed this test by changing the mapping rule specified for <Title> element. The changes performed are as follows:

SPARQL update query:

```
DELETE {
 GRAPH <http://xquery/XQrdf>
 { $s $p $o }
     }
INSERT {
 GRAPH <http://xquery/XQrdf>
 {$s $p 'principaltitleUpdate'}
```

```
    }

WHERE { $s $p $o
        FILTER regex ($o, 'principaltitle','i')
}
```

Expected outcome: We expected that the system should be successful in updating the triples through SPARQL and uplift the XML according to the updated mapping rule. The user should be able to locate the changes in the transformed RDF. In this particular test case the user should be able to locate the <principaltitleUpdate>...</principaltitleUpdate> element.

Actual outcome: User was able to successfully update the triples through SPARQL query. The transformation performed after updating the mapping was completed without any exception. The <principaltitleUpdate>...</principaltitleUpdate> element has been located by the user in the transformed output.

6.1.7 Test case 7: Changing mapping by updating triples-II

Purpose: The purpose of this is to test the capability of the system to support the user in changing the mapping rules. The user is changing the mapping rules by updating triples of "XQuery RDF" (base URI: http://xquery/XQrdf) using SPARQL update queries. The user is then performing the transformation based on the updated mappings. In this particular test case the user is changing an attribute value in already defined mapping rule.

Execution: The user performed this test by changing the mapping rule specified for the <Title> element. The changes performed are as follows:

SPARQL update query:

```
DELETE {
 GRAPH <http://xquery/XQrdf>
 { $s $p $o }
     }
INSERT {
 GRAPH <http://xquery/XQrdf>
 {$s $p 'twice'}
     }
WHERE { $s $p $o
         FILTER regex ($s, 'attribute','i')
         FILTER regex ($o, 'yes','i')
}
```

Expected outcome: We expected that the system should be successful in updating the triples through SPARQL and uplift the XML according to the updated mapping rule. The user should be able to locate the changes in the transformed RDF. In this particular test case the user should be able to locate the attribute {update="twice"}.

Actual outcome: User was able to successfully update the triples through SPARQL query. The transformation performed after updating the mapping was completed without any exception. The attribute {update="twice"} has been located by the user in the transformed output.

6.1.8 Test case 8: Preserve blank node

Purpose: The application is by default configured to remove all the blank nodes from the transformed RDF. The purpose of this is to test the capability of the system to support the user in changing the configuration for preserving the blank node in transformed RDF. The user is changing the configuration and then performing the transformation based on the updated configurations. In this particular test case the user is preserving the blank node of <note/> element.

Execution: The user performed this test by changing the XML based configuration file. The changes performed are as follows:

Default configuration:

```
<root>
</root>
```

Updated configuration:

```
<root>
  <nodename>note</nodename>
</root>
```

Expected outcome: We expected that the system should be successful in handling the updated configurations and uplift the XML according to that. The user should be able to locate the changes in the transformed RDF. In this particular test case the user should be able to locate the blank node of <note> element.

Actual outcome: User was able to successfully update the configurations. The transformation performed after updating the configurations was completed without any exception. The blank node of <note> element has been located by the user in the transformed output.

6.1.9 Test case 9: Analyze the transformed RDF triples-I

Purpose: The purpose of this is to test the capability of the system to support the user in analyzing the transformed RDF triples. The user is executing SPARQL read queries in order to extract information out of the triples. In this particular test case the user is extracting the count of the resources created between the date "18000101" and "19000101" including these dates.

Execution: The user performed this test by executing the SPARQL read query over the RDF triples (Base URI: http://data.test.tcd.ie/resource). The query is as follows:

SPARQL update query:

```
PREFIX xsd:     <http://www.w3.org/2001/XMLSchema#>
SELECT $s (COUNT(?s) AS ?count)
FROM <http://data.test.tcd.ie/resource>
WHERE { $s $p $o
        FILTER regex ($p, 'resourceDateIssuedStart','i')
        FILTER (xsd:integer($o) > 18000101)
        FILTER (xsd:integer($o) < 19000101)
}
```

Expected outcome: We expected that the system should be successful in executing the SPARQL read query over the RDF triples. The user should be able to extract the information out of it. In this particular case the expected result is “8”.

Actual outcome: The SPARQL query has been successfully executed over the RDF triples. The extracted count is “8”.

6.1.10 Test case 10: Analyze the transformed RDF triples-II

Purpose: The purpose of this is to test the capability of the system to support the user in analyzing the transformed RDF triples. The user is executing SPARQL read queries in order to extract information out of the triples. In this particular test case the user is extracting the count of resources in which the Geographic location is "Rome"

Execution: The user performed this test by executing the SPARQL read query over the RDF triples (Base URI: http://data.test.tcd.ie/resource). The query is as follows:

SPARQL update query:

```
PREFIX xsd:    <http://www.w3.org/2001/XMLSchema#>
SELECT $s (COUNT(?s) AS ?count)
FROM <http://data.library.tcd.ie/resource>
WHERE { $s $p $o
        FILTER regex ($p, 'label','i')
        FILTER regex ($o, 'Rome', 'i')

}
```

Expected outcome: We expected that the system should be successful in executing the SPARQL read query over the RDF triples. The user should be able to extract the information out of it. In this particular case the expected result is “1”.

Actual outcome: The SPARQL query has been successfully executed over the RDF triples. The extracted count is “1”.

6.1.11 Test case 11: Changing mapping by updating triples-III

Purpose: The purpose of this is to test the capability of the system to support the user in changing the mapping rules. The user is changing the mapping rules by updating triples of “XQuery RDF” (base URI: http://xquery/XQrdf) using SPARQL update queries. The user is then performing the transformation based on the updated mappings. In this particular test case the user is adding a new element in already defined mapping rule.

Execution: The user performed this test by executing the SPARQL update query. The query is as follows:

SPARQL update query:

```
INSERT DATA {
GRAPH <http://xquery/XQrdf>
        {
        <http://xquery/DirElemConstructor810> <http://XQrdfname>  "newElement"
        }
}
```

Expected outcome: We expected that the system should be successful in updating the triples through SPARQL. The user then executes the transformation. Transformed RDF <u>should not</u> accommodate the changes made by the specified SPARQL query.

Actual outcome: User was able to successfully update the triples through SPARQL query. The transformation performed after updating the mapping was completed without any exception. The user did not locate any changes in the transformed RDF.

6.1.12 Test case 12: Changing mapping by updating triples-IV

Purpose: The purpose of this is to test the capability of the system to support the user in changing the mapping rules. The user is changing the mapping rules by updating triples of "XQuery RDF" (base URI: http://xquery/XQrdf) using SPARQL update queries. The user is then performing the transformation based on the updated mappings. In this particular test case the user is adding a new attribute in already defined mapping rule.

Execution: The user performed this test by executing the SPARQL update query. The query is as follows:

SPARQL update query:

```
INSERT DATA
```

```
{
GRAPH <http://xquery/XQrdf>
        {
        <http://xquery/attribute412> <http://XQrdfnewattribute>  "dummy"
        }
}
```

Expected outcome: We expected that the system should be successful in updating the triples through SPARQL. The user then executes the transformation. Transformed RDF **should not** accommodate the changes made by the specified SPARQL query.

Actual outcome: User was able to successfully update the triples through SPARQL query. The transformation performed after updating the mapping was completed without any exception. The user did not locate any changes in the transformed RDF.

6.2 Evaluation: User perspective

We conducted an experiment with multiple participants. The participants have been recruited through email/ social networks. We only recruited the participants who have the following characteristics:

- Older than 18 years.
- Student in the department of School of Computer Science and Statistics only.
- Have the experience of XML or RDF or triples or SPARQL.

The experiment is divided into two parts 1) Task and 2) Questionnaire. Both parts have been conducted on the same computer provided by the researcher.

Part 1: User has been asked to perform the tasks related to the change of mappings, which involved performing some operations (necessary instruction will be provided in order to carry out the operation) on the prototype application created for uplifting the data. Please refer the task set in the attached CD (\lifting\tasks\Taskset.docx).

Part 2: After completing with part 1, users have been asked to evaluate the prototype application by providing answers to questions which are based on the ease of using the application, time taken for completing the task and information provided for executing the task. Please refer the questionnaire in the attached CD (\lifting\questionnaire\Questionnaire.docx).

Figure 28 shows rating provided by the user in evaluating the system based on the ease of using the application.

Figure 29 shows rating provided by the user in evaluating the system based on the time taken for completing the task.

Figure 30 shows rating provided by the user in evaluating the system based on the information provided for executing the task.

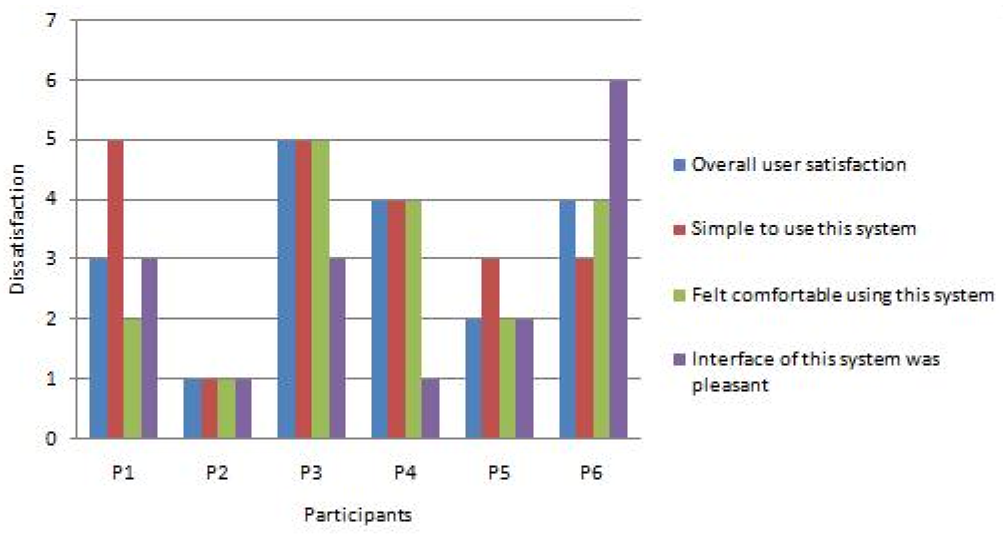


Figure 28: Ease of completing the task

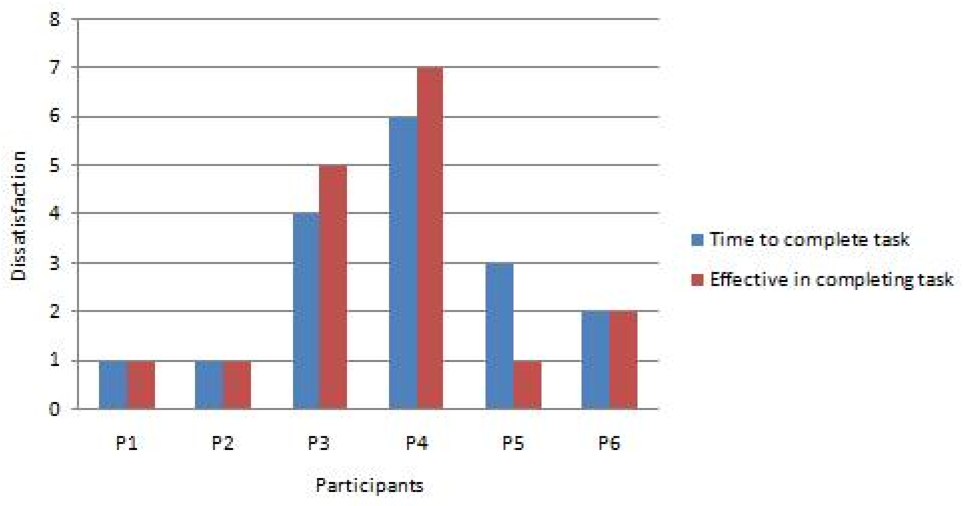


Figure 29: Time taken to complete the task

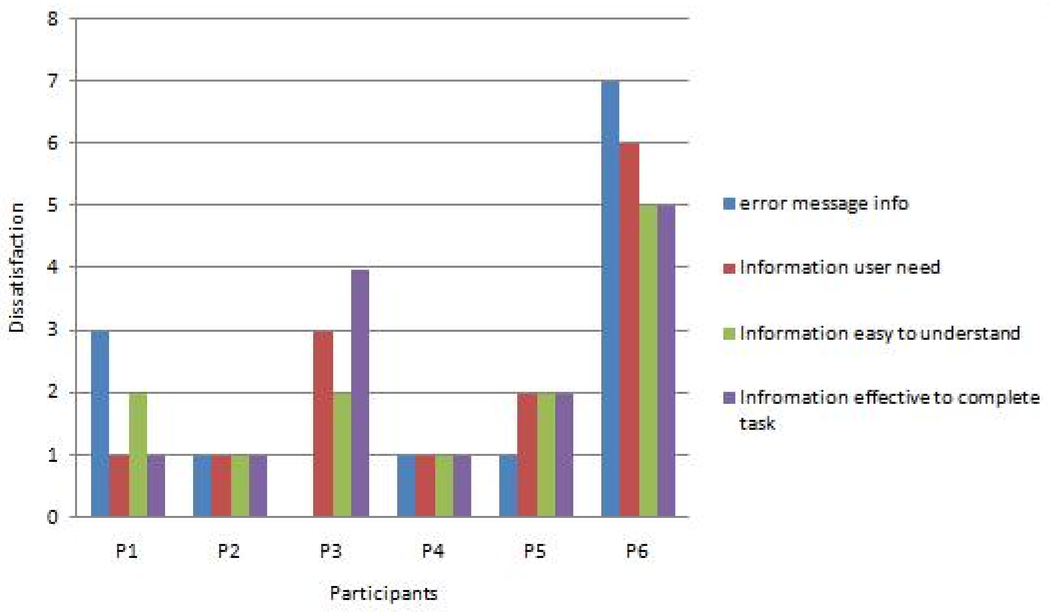


Figure 30: Support information

The information on the above graphs shows that, six users participated in the experiment. Participants 1, 2, 4, 5 and 6 are overall satisfied with the use of the application. According to participant 1 the guidelines provided for the execution of tasks were helpful in completing the task and it suggested that frequent use of the system will make the task easier to complete. Participant 1 also suggested that the error given by the system was very low level and hard to understand. Participant 1 found that the automated aspects of the system worked very well and made editing and managing the mappings very efficient. According to the ratings, Participant 2 looks very satisfied with the use of the application. The evaluation by participant 3 suggests that a more user friendly design is needed in place. It also suggests that participant 3 never encountered any error. Participant 4 indicates that it was unable to complete its task effectively, but it is satisfied with the error messages generated by the system. While the evaluation of participant 5 is completely opposite of participant 4, it indicates that it was able to complete its task very effectively. The tasks performed by the participant 4 and 5 were same; it suggests that the design of the system and information provided for executing the task were not compatible with every user. Participant 6 commented that the tasks were executed quickly and it can be quicker once the user become acquainted by the system. It commented that the error information generated by the system was not easily understandable by the user. Participant 6 also advised for putting more information on the interface such that the user can easily navigate through the application.

Based on the evaluation done by the user, it can be assimilated that a more user friendly design needs to be in place, and also the current prototype is unable to generate user understandable error messages.

6.3 Summary

In this chapter we have evaluated the developed prototype for uplifting the XML data. We have evaluated the prototype in following two areas:

Functionality of the prototype: We tested the functionality of the system with the help of unit test. We found that the test results are as expected.

Usability of the system: We evaluated the usability of the system by conducting the experiment to operate the prototype with multiple users. The operation involves tasks related to the change in already defined mapping.

Chapter 7

Case Studies

7.1 Introduction

In this chapter, we explore the application of the developed prototype on some industrial problems. We discuss the uplift of “Trinity library digital collection metadata” and “Employee dataset”. To test if the prototype is applicable in dealing with the real world problems of maintaining mappings, we also make changes in the mapping to see its effect.

7.2 Trinity library digital collection metadata

Over the past few years the library team has worked very hard in creating a digital collection of resources. In the process of digitizing the content, the team has to deal with the various format of metadata as different resources have different format of metadata. To make the digital collection more usable for the users, there should be interoperability between the related resources. This interoperability can be achieved effectively if we transform the metadata from multiple formats to a single format. The digital collection can be utilized at its best if uplift the metadata of resources into RDF. In this way we can ensure inter digital collection linking.

In this case study, we are uplifting the “Trinity library digital collection metadata” by the using the developed prototype. The uplift process has been carried out as follows:

- The process starts with the gathering of data. The Trinity library team provided us the data in FileMaker formatted XML. This XML contains the metadata of “583” records. Provided XML is very rich in terms of data, that is, it contains the information of every element used in representing the metadata information.

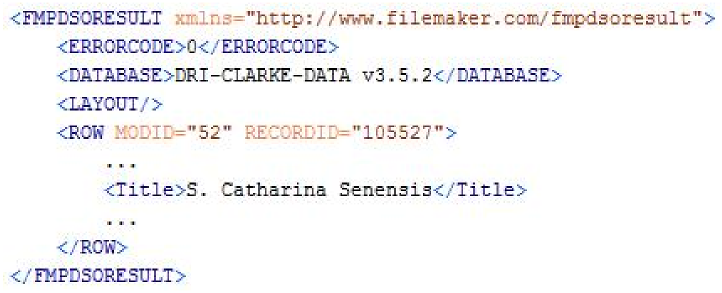
- The Trinity library team suggested that in their current workflow, the FileMaker data is transforming into the MODS format.

- The team wants the FileMaker data to be uplifted in the MODS RDF format in their new workflow. The team provided us the mapping information of FileMaker to MODS. The mapping information can be found in the attached CD (\lifting\assets\mappings\DRIS-preliminary-metadata-crosswalk.xlsx).

- We established the mapping of FileMaker to MODS XML to MODS RDF using the MODS XML to RDF conversion instructions [52]. The complete information about the mappings can be found in the attached CD (\lifting\assets\mappings\MappingInstructions.docx) as well.

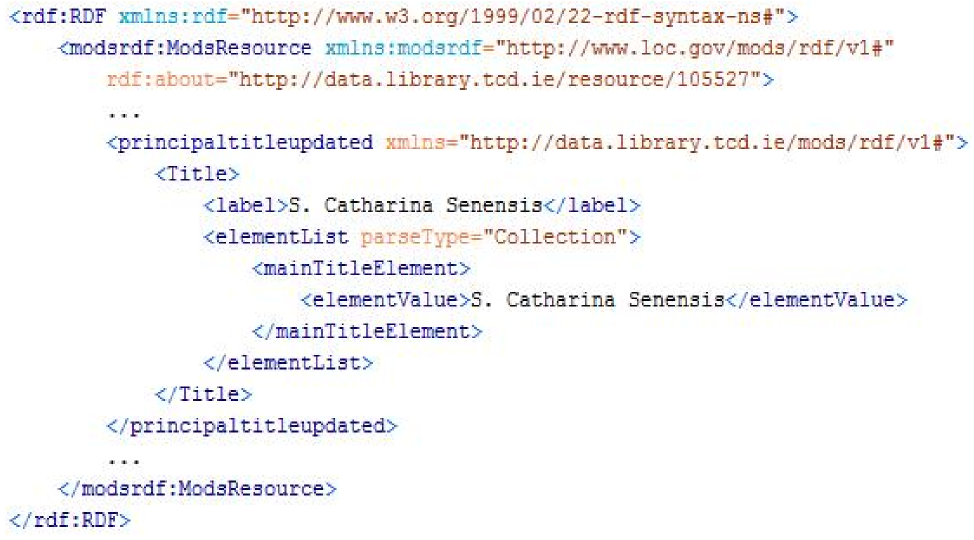
- We created an XQuery based on the mappings of MODS XML to RDF; this XQuery has been created using the conventions specified in section 5.2. The complete XQuery can be found in the attached CD (\lifting\assets\code\fileMakerToModsRDF.xquery).

- Once the XQuery has been created, the process of transformation can be initiated. The transformed MODS RDF can be found in the attached CD (\lifting\assets\modsRDF\modsRDF.rdf). **Figure 31** shows the title information of a resource in FileMaker format. After the transformation the FileMaker information has been converted to MODS RDF, **Figure 32** shows the title information in MODS RDF.



```
<FMPDSORESULT xmlns="http://www.filemaker.com/fmpdsoresult">
    <ERRORCODE>0</ERRORCODE>
    <DATABASE>DRI-CLARKE-DATA v3.5.2</DATABASE>
    <LAYOUT/>
    <ROW MODID="52" RECORDID="105527">
        ...
        <Title>S. Catharina Senensis</Title>
        ...
    </ROW>
</FMPDSORESULT>
```

Figure 31: Title information of a resource in FileMaker XML



```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
    <modsrdf:ModsResource xmlns:modsrdf="http://www.loc.gov/mods/rdf/v1#"
        rdf:about="http://data.library.tcd.ie/resource/105527">
        ...
        <principaltitleupdated xmlns="http://data.library.tcd.ie/mods/rdf/v1#">
            <Title>
                <label>S. Catharina Senensis</label>
                <elementList parseType="Collection">
                    <mainTitleElement>
                        <elementValue>S. Catharina Senensis</elementValue>
                    </mainTitleElement>
                </elementList>
            </Title>
        </principaltitleupdated>
        ...
    </modsrdf:ModsResource>
</rdf:RDF>
```

Figure 32: Title information of a resource in MODS RDF

- To test the prototype against the real world mapping challenges, we executed the tests mentioned in section 6.1. We found that the prototype was able to successfully manipulate the mappings through GUI as well as through SPARQL queries, details mentioned in section 6.1; however in the following to scenarios system was not able to update the mappings through SPARQL queries.
    - Adding a new element in already defined mapping rule.
    - Adding a new attribute in already defined mapping rule.

7.3 Employee dataset

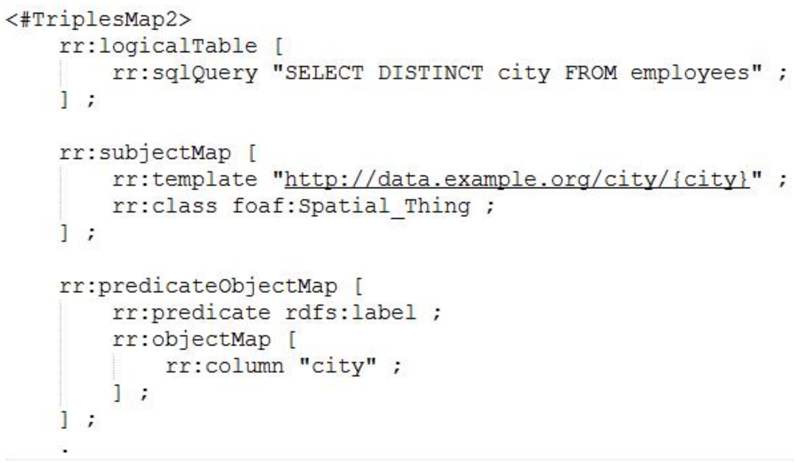
The second case study is based on the data which is more understandable to the audience who does not have understanding of publishing format. The purpose of this case study is to test the capability of the system to support various input XML formats to be uplifted. The process of its uplift has been carried out as follows.

- The process starts with the gathering of data. We have gathered this data from "Northwind database" in the XML format. The sample employee dataset can be found in the attached CD (\lifting\assets\Filemaker\employees.xml).

- The employee dataset needs to be uplifted into FOAF. FOAF is an ontology which is used in describing people, their activities, and their relation to other people.

- The mapping information of the employee dataset to FOAF has been provided by Dr. Christophe Debruyne. The provided mappings were in triple maps. We also received the desired outcome of the transformation for the validation of RDF generated by the prototype.

- We converted the "triple mappings" in XQuery. Most of the mappings were straight forward except mapping of the city (employee dataset) to foaf:Spatial_Thing (FOAF). The mapping in the triple map is indicating to cater only distinct city information. The complete mapping can be found in the attached CD (\lifting\assets\mappings\e-mapping.ttl). We managed to convert the distinct condition with the help of XPATH axes in which we checked if the city has already occurred then there is no need for its conversion again. **Figure 33** and **Figure 34** show mapping information in triple map and XQuery respectively.

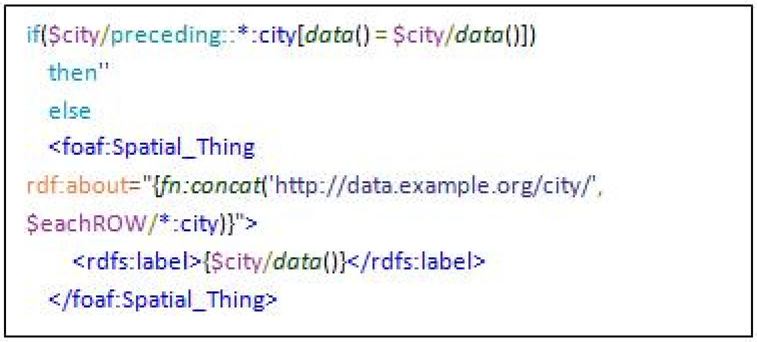


```
<#TriplesMap2>
    rr:logicalTable [
        rr:sqlQuery "SELECT DISTINCT city FROM employees" ;
    ] ;

    rr:subjectMap [
        rr:template "http://data.example.org/city/{city}" ;
        rr:class foaf:Spatial_Thing ;
    ] ;

    rr:predicateObjectMap [
        rr:predicate rdfs:label ;
        rr:objectMap [
            rr:column "city" ;
        ] ;
    ] ;
    .
```

Figure 33: mapping of city in triple maps



```
if($city/preceding::*:city[data() = $city/data()])
  then''
  else
  <foaf:Spatial_Thing
rdf:about="{fn:concat('http://data.example.org/city/',
$eachROW/*:city)}">
     <rdfs:label>{$city/data()}</rdfs:label>
  </foaf:Spatial_Thing>
```

Figure 34: Mapping of city in XQuery

- We converted all the mappings in XQuery. The complete XQuery can be found in the attached CD (\lifting\assets\code\Foaf.xqy).

- Once the XQuery has been created, the process of transformation can be initiated. We compared the RDF generated by the prototype with turtle received as the expected outcome. We found that both of the outputs are equivalent. Both of the outputs can be found in the attached CD (\lifting\assets\modsRDF\e-output.ttl) and (\lifting\assets\modsRDF\FOAF.rdf).

- This case study suggests that the developed prototype is compatible with different datasets to be uplifted.

Chapter 8

Conclusion

This chapter describes the summary of the research and our findings. In the later section the future work has also been discussed.

8.1 Project overview

Data is nowadays embedded in our day to day activities and different domains, such as hospitals, entertainment, schools and so on, use different formats for the data. With the growth of the Semantic Web and linked data initiatives the challenge of extracting information from different domains and transforming the data to a standard format (RDF) exists.

There are solutions available in the current state of the art, which performs the mapping of one format to another to perform the extraction and transformation. However, in case any of the format gets changed, then these mapping needs to be established again and further the transformation logic has to be adjusted accordingly [38].

The most recent developments in the field of mapping are [10], [20] and [28]. These solutions perform the mapping of various format (relational databases, XML, CSV, etc.) to RDF. But these approaches only focused on the generic solution for the transformation of one format to RDF. None of the solution has targeted the representation of the mappings. This has provided us an area where we can discuss the representation of mappings in order to analyze, manipulate and recompose them automatically.

For this project, our aim is to represent the mappings (embedded in XQuery) into the RDF structure. As RDF is the machine understandable language, it can be analyzed, manipulated and recomposed automatically. This approach will be beneficial in the domain where the changes in the mappings are frequent. These changes lead the maintenance of the mapping a very challenging task.

We propose a prototype which is implemented using MarkLogic [41]. This prototype transforms the input XML to RDF using mappings embedded in XQuery. We are also converting the XQuery to RDF such that it can be analyzed and manipulated automatically. The advantages of keeping the mappings/ XQuery in RDF are as follows:

- With the help of a transformation, it allows us to render the mappings in user readable format. The user can then edit the mappings manually.

- It allows us to store the triples in the triple store. One can automatically analyze the mappings or transformation code using the SPARQL queries. In this case mappings can be changed automatically by updating the triples using the SPARQL update queries.

We are recomposing XQuery after the update in mappings using the round-trip engine. This engine is transforming “XQuery RDF” back to “XQuery expressions“.

Finally, we have evaluated the developed prototype by running the test cases of the following functionality:

- Manually update the already defined mappings using GUI.

- Automatically update the already defined mappings using SPARQL update queries.

- Analyze the mapping/ transformation code using SPARQL read queries.

- Analyze the transformed RDF of input XML for extracting useful information.

- By default the application is configured to remove all the blank nodes, we tested for preserving the blank node by changing the configuration of the application.

For testing the above cases we have used the Filemaker dataset, which needs to be uplifted to modsRDF.

8.2 Contribution

In summary, our prototype validates the approach of uplifting the data using the XQuery. It also validates that the mappings can be represented in a format such that it can be automatically analyzed, manipulated and recomposed. Or evaluation result illustrates that such an approach can be used to uplift the data. It also proved that by representing the mappings in the RDF one can easily cope up with the maintenance of the mappings.

Furthermore, our prototype demonstrates the approach to transforming the XQuery to RDF in a way that it can be recomposed back to the XQuery. To conclude, this research has successfully defined an approach by which mappings can be represented using a query based language such that it can be analyzed, manipulated and recomposed automatically.

8.3 Future work

While our prototype has represented the mappings in RDF structure and allows the user to update the already defined mapping, the future extension to the system might be the functionality by which user can add the mappings of the new elements.

Another possible extension to the system would be to render the mappings in a more user friendly way. In the current rendering scheme the user should have an exposure of XML and XQuery in order to update the mappings. As XQuery is not a common language at application operator level, the XQuery snippets in the rendered mapping should be replaced with something closer to the users.

Finally, as in this, research we have created the prototype and tested this with two different input dataset, the future work may be done to employ this prototype with more input dataset. This would help to validate the approach taken in this research, by using it to represent the more complex mappings.

8.4 Final remark

People are moving more towards the Semantic Web. As a result, they are uplifting their data in RDF. However the schemas or ontologies of the data tend to change frequently, the need for an approach has arisen by which mappings can be automatically analyzed and manipulated. This research has successfully defined an approach by which the mapping can be represented in the RDF format, which can be automatically analyzed, manipulated and recomposed. The overall inspiration for this study is the representation of XQuery in the RDF format which allows the further reasoning over the XQuery itself.

References

[1] Bizer, Christian and Heath, Tom and Berners-Lee, Tim. "Linked data-the story so far". W3C, 5(3):205-227, 2009

[2] "Semantic web road map." [Online]. Available: https://www.w3.org/DesignIssues/Semantic.html. [Modified: 14 Oct 1998]. [Accessed: 11 Mar 2016].

[3] Neuhold, Erich and Kiesling, Elmar. "Interoperability and Semantics: A Never-Ending Story". Information Sciences and Systems 2015, Springer, 19-31, 2016

[4] "RDF - Semantic Web Standards" [Online]. Available: http://www.w3.org/RDF/. [Modified: 17 Feb 2016]. [Accessed: 14 Jul 2016].

[5] Miller, Eric. "An introduction to the resource description framework". Bulletin of the American Society for Information Science and Technology, 25(1):15-19, 1998

[6] "RDF/XML syntax specification (revised)" [Online]. Available: http://badjoke.demic.eu/papers/liris.cnrs.fr/alain.mille/enseignements/Ecole_Centrale/projets_2004/RDF.pdf. [Modified: 10 Oct 2003]. [Accessed: 14 Jul 2016].

[7] "Turtle - Terse RDF Triple Language" [Online]. Available: https://www.w3.org/TeamSubmission/turtle/. [Modified: 28 March 2011]. [Accessed: 14 Jul 2016].

[8] "SPARQL Query Language for RDF" [Online]. Available: https://www.w3.org/TR/rdf-sparql-query/. [Modified: 15 Jan 2008]. [Accessed: 14 Jul 2016].

[9] Marinchev, Ivo. "Lifting and lowering the data from digital library Virtual Encyclopedia of Bulgarian Iconography". Proceedings of the 12th International Conference on Computer Systems and Technologies, pages 179-184, Jun, 2011

[10] Bizer, Christian. "D2r map-a database to rdf mapping language". 2003

[11] Hendler, James and Brners-Lee, T and Miller, Eric. "Integrating applications on the semantic web". Journal of the Institute of Electrical Engineers of Japan, 122(10):676-680, 2002

[12] Dimou, Anastasia and Vander Sande, Miel and Colpaert, Pieter and Verborgh, Ruben and Mannens, Erik and Van de Walle, Rik. "RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data". LDOW, Apr 2014.

[13] Lopes, Nuno and Polleres, Axel and Passant, Alexandre and Decker, Stefan and Bischof, Stefan and Berrueta, Diego and Campos, Antonio and Corlosquet, Stephane and Euzenat, Jerome and Erling, Orri. "RDF and XML: Towards a unified query layer". Proceedings of the W3C Workshop on RDF Next Steps, Jun 2010.

[14] Bizer, Christian and Seaborne, Andy. "D2RQ-treating non-RDF databases as virtual RDF graphs". Proceedings of the 3rd international semantic web conference (ISWC2004), Nov 2004, Springer.

[15] Van Deursen, Davy and Poppe, Chris and Martens, Gaetan and Mannens, Erik and Van de Walle, Rik. "XML to RDF conversion: a generic approach". Automated solutions for Cross Media Content and Multi-channel Distribution, 2008. AXMEDIS'08. International Conference on, pages 138-144, Nov 2008, IEEE.

[16] Dimou, Anastasia and Sande, Miel Vander and Colpaert, Pieter and Mannens, Erik and Van de Walle, Rik. "Extending R2RML to a source-independent mapping language for RDF". Proceedings of the 2013th International Conference on Posters \ Demonstrations Track-Volume 1035, pages 237-240, Oct 2013, CEUR-WS. org.

[17] Ferdinand, Matthias and Zirpins, Christian and Trastour, David. "Lifting XML schema to OWL". International Conference on Web Engineering, pages 354-358, Jul 2004, Springer.

[18] Ehrig, Marc and Sure, York. "Ontology mapping--an integrated approach". European Semantic Web Symposium, pages 76-91, May 2004, Springer.

[19] "RDF Mapping Language (RML)" [Online]. Available: http://semweb.datasciencelab.be/rml/RMLmappingLanguage.html. [Accessed: July 22 2016].

[20] Dimou, Anastasia and Vander Sande, Miel and Slepicka, Jason and Szekely, Pedro and Mannens, Erik and Knoblock, Craig and Van de Walle, Rik. "Mapping hierarchical sources into RDF using the RML mapping language". Semantic Computing (ICSC), 2014 IEEE International Conference on, pages 151-158, Jun 2014, IEEE.

[21] Scharffe, Francois and Atemezing, Ghislain and Troncy, Raphael and Gandon, Fabien and Villata, Serena and Bucher, Benedicte and Hamdi, Faycal and Bihanic, Laurent and Kepeklian, Gabriel and Cotton, Franck. "Enabling linked-data publication with the datalift platform". Proc. AAAI workshop on semantic cities, Jul 2012, AAAI.

[22] David, J and Euzenat, J and Scharffe, F and dos Santos, C Trojahn. "The Alignment API 4.0". Semantic Web-Interoperability, Usability, Applicability, 2(1):3-10, 2011.

[23] Euzenat, Jerome. "An API for ontology alignment". International Semantic Web Conference, pages 698-712, Nov 2004, Springer.

[24] "EDOAL: Expressive and Declarative Ontology Alignment Language" [Online]. Available: http://alignapi.gforge.inria.fr/edoal.html. [Accessed: 22 July 2016].

[25] "Alignments for data interlinking: a proposal for matching/linking cooperation" [Online]. Available: http://melinda.inrialpes.fr/proposal.html. [Accessed: 22 July 2016].

[26] Kurgan, Lukasz A and Swiercz, Waldemar and Cios, Krzysztof J. "Semantic Mapping of XML Tags Using Inductive Machine Learning". ICMLA, pages 99-109, Jun 2002, CSREA.

[27] Sahoo, Satya S and Halb, Wolfgang and Hellmann, Sebastian and Idehen, Kingsley and Thibodeau Jr, Ted and Auer, Soren and Sequeda, Juan and Ezzat, Ahmed. "A survey of current approaches for mapping of relational databases to RDF". W3C RDB2RDF Incubator Group Report, 2009.

[28] Dell'Aglio, Daniele and Polleres, Axel and Lopes, Nuno and Bischof, Stefan. "Querying the web of data with XSPARQL 1.1". Proceedings of the 2014 International Conference on Developers-Volume 1268, pages 113-118, Oct 2014, CEUR-WS. org.

[29] Bischof, Stefan and Decker, Stefan and Krennwallner, Thomas and Lopes, Nuno and Polleres, Axel. "Mapping between RDF and XML with XSPARQL". Journal on Data Semantics, 1(3):147-185, 2012.

[30] Akhtar, Waseem and Kopecky, Jacek and Krennwallner, Thomas and Polleres, Axel. "XSPARQL: Traveling between the XML and RDF worlds--and avoiding the XSLT pilgrimage". European Semantic Web Conference, pages 432-447, Jun 2008, Springer.

[31] Carlisle, David. "xq2xml: Transformations on XQueries". a conference on XML, pages 63, Jun 2006.

[32] Lopes, Nuno and Bischof, Stefan and Decker, Stefan and Polleres, Axel. "On the semantics of heterogeneous querying of relational, XML and RDF data with XSPARQL". Proceedings of the 15th Portuguese Conference on Artificial Intelligence, Oct 2011, EPIA.

[33] "XQuery 1.0 and XPath 2.0 Grammar Test Pages Overview" [Online]. Available: https://www.w3.org/2007/01/applets/xgrammar.zip. [Accessed: July 22 2016].

[34] Milo, Tova and Zohar, Sagit. "Using schema matching to simplify heterogeneous data translation". VLDB, pages 24-27, Aug 1998, Citeseer.

[35] Marshall, M Scott and Boyce, Richard and Deus, Helena F and Zhao, Jun and Willighagen, Egon L and Samwald, Matthias and Pichler, Elgar and Hajagos, Janos and Prud'hommeaux, Eric and Stephens, Susie. "Emerging practices for mapping and linking life sciences data using RDF—A case series". Web Semantics: Science, Services and Agents on the World Wide Web, 14, 2-13, 2012.

[36] Battle, Steve. "Round-tripping between XML and RDF". International Semantic Web Conference (ISWC), Nov 2004, Citeseer.

[37] Harth, Andreas and Hogan, Aidan and Umbrich, Jurgen and Decker, Stefan. "Building a semantic web search engine: challenges and solutions". Proc. of XTech 2008:"The Web on the Move, 2008.

[38] Nagarajan, Meenakshi and Verma, Kunal and Sheth, Amit P and Miller, John and Lathem, Jon. "Semantic interoperability of web services-challenges and experiences".

2006 IEEE International Conference on Web Services (ICWS'06), pages 373-382, Sep 2006, IEEE.

[39] Rimkus, Kyle and Padilla, Thomas and Popp, Tracy and Martin, Greer. "Digital preservation file format policies of ARL member libraries: an analysis". D-Lib Magazine, 20(3):2, 2014.

[40] Baker, Thomas and Dekkers, Makx and Heery, Rachel and Patel, Manjula and Salokhe, Gauri. "What terms does your metadata use? Application profiles as machine-understandable narratives". Journal of Digital information, 2(2):2001.

[41] "Inside MarkLogic Server" [Online]. Available: http://www.marklogic.com/resources/inside-marklogic-server/. [Accessed: Aug 10 2016].

[42] "Extensible Markup Language (XML)" [Online]. Available: http://www.w3.org/XML/. [Accessed: Aug 10 2016].

[43] Chamberlin, Don. "XQuery: An XML query language". IBM systems journal, 41(4):597-615, 2002.

[44] Derksen, Eduard and Fankhauser, Peter and Howland, Ed and Huck, Gerald and Macherius, Ingo and Murata, Makoto and Resnick, Michael and Schoning, Harald. "XQL (XML Query Language)". Submission to the World Wide Web Consortium, 1999.

[45] Clark, James and DeRose, Steve. "XML path language (XPath) version 1.0". 184-186, 1999.

[46] Krishnamurthy, Rajasekar and Kaushik, Raghav and Naughton, Jeffrey F. "XML-to-SQL query translation literature: The state of the art and open problems". International XML Database Symposium, pages 1-18, Sep 2003, Springer.

[47] Cattell, Roderic Geoffrey Galton and Barry, Douglas K and Bartels, Dirk and Berler, Mark and Eastman, Jeff and Gamerman, Sophie and Jordan, David and Springer, Adam and Strickland, Henry and Wade, Drew. "The object database standard". ODMG 2.0, Morgan Kaufmann Publishers San Mateo, 1997.

[48] "XSL Transformations (XSLT) Version 2.0" [Online]. Available: http://www.w3.org/TR/xslt20. [Accessed: 14 Jul 2016].

[49] Westbrooks, Elaine L. "Distributing and synchronizing heterogeneous metadata in geospatial information repositories for access". 2004.

[50] "XML document" [Online]. Available: https://www.w3.org/2007/01/applets/xpath-grammar.xml. [Accessed: 14 Jul 2016].

[51] "Northwind database" [Online]. Available: https://github.com/dalers/mywind. [Accessed: 10 Aug 2016].

[52] "MODS XML to RDF Conversion" [Online]. Available: https://www.loc.gov/standards/mods/modsrdf/primer-2.html. [Accessed: 10 Aug 2016].

[53] Dimou, Anastasia and Vander Sande, Miel and Slepicka, Jason and Szekely, Pedro and Mannens, Erik and Knoblock, Craig and Van de Walle, Rik. "Mapping hierarchical sources into RDF using the RML mapping language". Semantic Computing (ICSC), 2014 IEEE International Conference on, pages 151-158, Jun 2014, IEEE.

Appendix A

Intermediate XML Elements	XSLT rules to generated “XQuery RDF”
CompElemConstructor	<xsl:element name="XQrdf:compelement"> <xsl:attribute name="rdf:about" select="concat($namespace_string, local-name())"/> <xsl:apply-templates/> </xsl:element>
ComparisonExpr	<xsl:element name="XQrdf:compare"> <xsl:attribute name="rdf:about" select="concat($namespace_string, 'compare')"/> <xsl:attribute name="XQrdf:value" select="normalize-space(text())"/> <xsl:apply-templates/> </xsl:element>
DirAttributeList	<xsl:apply-templates select="TagQName"/>
DirElemConstructor	<xsl:element name="XQrdf:element"> <xsl:attribute name="rdf:about" select="concat($namespace_string, local-name())"/> <xsl:attribute name="XQrdf:name" select="normalize-space(TagQName)"/> <xsl:apply-templates/> </xsl:element>
FLWORExpr	<xsl:element name="XQrdf:FLWORExpr"> <xsl:attribute name="rdf:about" select="concat($namespace_string, local-name())"/> <xsl:apply-templates/> </xsl:element>
ForClause	<xsl:element name="XQrdf:for"> <xsl:attribute name="rdf:about" select="concat($namespace_string, local-name())"/> <xsl:apply-templates/> </xsl:element>
FunctionCall	<xsl:element name="XQrdf:function"> <xsl:attribute name="rdf:about" select="concat($namespace_string, local-name())"/> <xsl:attribute name="XQrdf:name"> <xsl:variable name="val" select="normalize-space(FunctionQName)"/> <xsl:value-of select="$val"/> </xsl:attribute> <xsl:apply-templates/> </xsl:element>
IfExpr	<xsl:element name="XQrdf:if"> <xsl:attribute name="rdf:about" select="concat($namespace_string, local-name())"/> <xsl:apply-templates/> </xsl:element>

IntegerLiteral	<xsl:element name="XQrdf:integer">
	<xsl:attribute name="rdf:about" select="concat($namespace_string, local-name())"/>
	<xsl:attribute name="XQrdf:value" select="normalize-space(.)"/>
	</xsl:element>
LetClause	<xsl:element name="XQrdf:let">
	<xsl:attribute name="rdf:about" select="concat($namespace_string, local-name())"/>
	<xsl:message select="VarName"/>
	<xsl:apply-templates/>
	</xsl:element>
NamespaceDecl	<xsl:element name="XQrdf:namespace">
	<xsl:attribute name="rdf:about" select="concat($namespace_string, 'namespace')"/>
	<xsl:attribute name="XQrdf:name" select="normalize-space(NCName)"/>
	<xsl:attribute name="XQrdf:value" select="normalize-space(URILiteral/StringLiteral)"/>
	</xsl:element>
PathExpr	<xsl:choose>
	<xsl:when test="parent::*:FLWORExpr">
	<xsl:element name="XQrdf:return">
	<xsl:attribute name="rdf:about" select="concat($namespace_string, 'Constructor')"/>
	<xsl:apply-templates/>
	</xsl:element>
	</xsl:when>
	<xsl:otherwise>
	<xsl:element name="XQrdf:expression">
	<xsl:attribute name="rdf:about" select="concat($namespace_string, local-name())"/>
	<xsl:apply-templates/>
	</xsl:element>
	</xsl:otherwise>
	</xsl:choose>
PositionalVar	<xsl:element name="XQrdf:position">
	<xsl:attribute name="rdf:about" select="concat($namespace_string, local-name())"/>
	<xsl:apply-templates/>
	</xsl:element>
Predicate	<xsl:element name="XQrdf:predicate">
	<xsl:attribute name="rdf:about" select="concat($namespace_string, local-name())"/>
	<xsl:apply-templates/>
	</xsl:element>
PredicateList	<xsl:element name="XQrdf:predicatelist">
	<xsl:attribute name="rdf:about" select="concat($namespace_string, local-name())"/>
	<xsl:apply-templates/>
	</xsl:element>

QName[parent::NameTest and not(ancestor::AbbrevForwardStep[1]/text() = '@')]	<xsl:element name="XQrdf:node"> <xsl:attribute name="rdf:about" select="concat($namespace_string, 'StarColonNCName')"/> <xsl:attribute name="XQrdf:value" select="normalize-space(.)"/> </xsl:element>
QName[parent::NameTest/ ancestor::AbbrevForwardStep[1]/text() = '@']	<xsl:element name="XQrdf:node"> <xsl:attribute name="rdf:about" select="concat($namespace_string, 'StarColonNCName')"/> <xsl:attribute name="XQrdf:value" select="concat('@', normalize-space(.))"/> </xsl:element>
QueryList	<xsl:apply-templates/>
ReverseAxis	<xsl:element name="XQrdf:axis"> <xsl:attribute name="rdf:about" select="concat($namespace_string, 'axis')"/> <xsl:attribute name="XQrdf:value" select="normalize-space(concat(text(), '::'))"/> <xsl:apply-templates/> </xsl:element>
SlashSlash	<xsl:element name="XQrdf:doubleslash"> <xsl:attribute name="rdf:about" select="concat($namespace_string, local-name())"/> <xsl:apply-templates/> </xsl:element>
StarColonNCName	<xsl:element name="XQrdf:node"> <xsl:attribute name="rdf:about" select="concat($namespace_string, local-name())"/> <xsl:attribute name="XQrdf:value"> <xsl:variable name="val" select="normalize-space(.)"/> <xsl:value-of select="$val"/> </xsl:attribute> <xsl:apply-templates/> </xsl:element>
StepExpr	<xsl:element name="XQrdf:step"> <xsl:attribute name="rdf:about" select="concat($namespace_string, local-name())"/> <xsl:apply-templates/> </xsl:element>
StringLiteral[normalize-space(.) != '']	<xsl:element name="XQrdf:string"> <xsl:attribute name="rdf:about" select="concat($namespace_string, local-name())"/> <xsl:attribute name="XQrdf:value"> <xsl:variable name="val" select="normalize-space(.)"/> <xsl:value-of select="$val"/> </xsl:attribute> <xsl:apply-templates/> </xsl:element>

TagQName	<xsl:choose>
	  <xsl:when test="parent::DirAttributeList">
	   <xsl:element name="XQrdf:attribute">
	    <xsl:attribute name="rdf:about" select="concat($namespace_string, 'attribute')"/>
	    <xsl:attribute name="XQrdf:name" select="normalize-space(.)"/>
	    <xsl:choose>
	     <xsl:when test="following-sibling::DirAttributeValue[1]/descendant::Lbrace">
	      <xsl:apply-templates select="following-sibling::DirAttributeValue[1]/*"/>
	     </xsl:when>
	     <xsl:otherwise>
	      <xsl:attribute name="XQrdf:value">
	       <xsl:variable name="val">
	        <xsl:for-each select="following-sibling::DirAttributeValue[1]/*">
	         <xsl:value-of select="normalize-space(.)"/>
	        </xsl:for-each>
	       </xsl:variable>
	       <xsl:value-of select="normalize-space($val)"/>
	      </xsl:attribute>
	     </xsl:otherwise>
	    </xsl:choose>
	   </xsl:element>
	  </xsl:when>
	 </xsl:choose>

VarName	<xsl:element name="XQrdf:varname">
	  <xsl:attribute name="rdf:about" select="concat($namespace_string, local-name())"/>
	  <xsl:attribute name="XQrdf:value">
	   <xsl:variable name="val" select="normalize-space(QName)"/>
	   <xsl:value-of select="$val"/>
	  </xsl:attribute>
	  <xsl:apply-templates/>
	 </xsl:element>

Appendix B

"XQuery RDf" Elements	XSLT rules to generate "Xquery statements"
Axis	`<xsl:value-of select="@XQrdf:value"/>`

compelement	`<xsl:text> element {</xsl:text>`
	`<xsl:apply-templates select="*:expression[1]"/>`
	`<xsl:text>} {</xsl:text>`
	`<xsl:apply-templates select="*:expression[2]"/>`
	`<xsl:text>}</xsl:text>`

doubleslash	`<xsl:choose>`
	`<xsl:when test="preceding-sibling::*[1][local-name() = 'varname'] and following-sibling::*[1][local-name() = 'function']"/>`
	`<xsl:otherwise>`
	`<xsl:text>//</xsl:text>`
	`</xsl:otherwise>`
	`</xsl:choose>`

element

```
<xsl:when test="child::*:element">
   <xsl:element name="{@XQrdf:name}">
    <xsl:for-each select="*:attribute[not(starts-with(@*:name,
'xmlns'))]">
     <xsl:attribute name="{@XQrdf:name}">
      <xsl:variable name="val">
       <xsl:choose>
        <xsl:when test="child::*">
         <xsl:text>{</xsl:text>
         <xsl:apply-templates mode="attributeval"/>
         <xsl:text>}</xsl:text>
        </xsl:when>
        <xsl:otherwise>
         <xsl:value-of select="@*:value"/>
        </xsl:otherwise>
       </xsl:choose>
      </xsl:variable>
      <xsl:value-of select="$val"/>
     </xsl:attribute>
    </xsl:for-each>
    <xsl:apply-templates/>
   </xsl:element>
  </xsl:when>
  <xsl:otherwise>
   <xsl:element name="{@*:name}">
    <xsl:for-each select="*:attribute[not(starts-with(@*:name,
'xmlns'))]">
     <xsl:attribute name="{@*:name}">
      <xsl:variable name="val">
       <xsl:choose>
        <xsl:when test="child::*">
         <xsl:text>{</xsl:text>
         <xsl:apply-templates mode="attributeval"/>
         <xsl:text>}</xsl:text>
        </xsl:when>
        <xsl:otherwise>
         <xsl:value-of select="@*:value"/>
        </xsl:otherwise>
       </xsl:choose>
      </xsl:variable>
      <xsl:value-of select="$val"/>
     </xsl:attribute>
    </xsl:for-each>
    <xsl:variable name="check">
     <xsl:apply-templates/>
    </xsl:variable>
    <xsl:choose>
     <xsl:when test="normalize-space($check) != ''">
      <xsl:text>{</xsl:text>
      <xsl:copy-of select="$check"/>
      <xsl:text>}</xsl:text>
     </xsl:when>
     <xsl:otherwise>
      <xsl:copy-of select="$check"/>
     </xsl:otherwise>
    </xsl:choose>
   </xsl:element>
  </xsl:otherwise>
 </xsl:choose>
```

expression	<xsl:choose>
	<xsl:when test="parent::*:attribute"/>
	<xsl:when test="parent::*:compare and preceding-sibling::*[1][self::*:expression]">
	<xsl:value-of select="concat(' ', parent::*:compare/@*:value, ' ')"/>
	<xsl:apply-templates/>
	</xsl:when>
	<xsl:otherwise>
	<xsl:if test="preceding-sibling::*[1][self::*:expression] and not(parent::*:if) and not(parent::*:compelement)">
	<xsl:choose>
	<xsl:when test="parent::XQrdf:compare">
	<xsl:value-of select="parent::XQrdf:compare/@XQrdf:value"/>
	</xsl:when>
	<xsl:otherwise>
	<xsl:text>, </xsl:text>
	</xsl:otherwise>
	</xsl:choose>
	</xsl:if>
	<xsl:apply-templates/>
	</xsl:otherwise>
	</xsl:choose>
for	<xsl:choose>
	<xsl:when test="child::*:position">
	<xsl:value-of select="concat('for $', *:varname/@*:value)"/>
	</xsl:when>
	<xsl:otherwise>
	<xsl:value-of select="concat('for $', *:varname/@*:value, ' in ')"/>
	</xsl:otherwise>
	</xsl:choose>
	<xsl:apply-templates/>
function	<xsl:if test="preceding-sibling::*:varname">
	<xsl:text>/</xsl:text>
	</xsl:if>
	<xsl:value-of select="concat(@*:name, '(')"/>
	<xsl:apply-templates/>
	<xsl:text>)</xsl:text>

if	<xsl:if test="preceding-sibling::*[1][self::*:let]"> <xsl:text> return </xsl:text> </xsl:if> <xsl:text> if(</xsl:text> <xsl:apply-templates select="*:expression[1]"/> <xsl:text>) then</xsl:text> <xsl:apply-templates select="*:expression[2]"/> <xsl:text> else </xsl:text> <xsl:apply-templates select="*[3]"/>
integer	<xsl:value-of select="normalize-space(@*:value)"/>
let	<xsl:value-of select="concat('let $', *:varname/@*:value, ' := ')"/> <xsl:apply-templates/>
node	<xsl:if test="matches(normalize-space(@*:value), '[a-zA-Z]')"> <xsl:value-of select="normalize-space(@*:value)"/> </xsl:if>
position	<xsl:value-of select="concat(' at $', *:varname/@*:value, ' in ')"/>
predicate	<xsl:text>[</xsl:text> <xsl:apply-templates/> <xsl:text>]</xsl:text>
return	<xsl:text> return</xsl:text>(<xsl:apply-templates/>)
step	<xsl:choose> <xsl:when test="preceding-sibling::*[1][self::*:doubleslash]"/> <xsl:when test="normalize-space(*:node/@*:value) = ''"/> <xsl:when test="not(parent::*:expression/preceding-sibling::*) and parent::*:expression/*:compare"/> <xsl:otherwise> <xsl:text>/</xsl:text> </xsl:otherwise> </xsl:choose> <xsl:apply-templates/>
string	<xsl:value-of select="normalize-space(@*:value)"/>

varname	<xsl:choose>
	  <xsl:when test="parent::*:expression">
	   <xsl:value-of select="concat('$', @*:value)"/>
	  </xsl:when>
	  <xsl:when test="parent::*:return">
	   <xsl:text>
	        </xsl:text>
	   <xsl:value-of select="concat('$', @*:value)"/>
	   <xsl:text>
	        </xsl:text>
	  </xsl:when>
	 </xsl:choose>
	 <xsl:if test="preceding-sibling::*[1][ends-
	with(self::*:step/*:node/@*:value, '[')]">
	  <xsl:text>]</xsl:text>
	 </xsl:if>

expressionattributeval	<xsl:if test="preceding-sibling::*[1][self::*:expression]"> <xsl:text>, </xsl:text> </xsl:if> <xsl:apply-templates/>
namespace	<xsl:namespace name="{@*:name}" select="@*:value"/>