TOWARDS AUTONOMIC UPLIFT OF INFORMATION

Sarah Alzahrani

A dissertation submitted to the University of Dublin

in partial fulfillment of the requirements for the degree of

Master of Science in Computer Science

2015

Declaration

I 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.

Signed: ___________________

Sarah Alzahrani

28th August 2015

Permission to lend and/or copy

Acknowledgments

I would especially like to thank my supervisor, Prof. Declan O'Sullivan, for having proposed this topic and for all his guidance and assistance throughout the duration of the project. I would like to thank Alan Meehan, a PhD student from the Knowledge and Data Engineering group for sharing his design. I also wish to thank my family and friends for their love and support during this year. Special thanks and appreciation goes to my father for his endless support and for being here with me through this experience. Finally, I would like to express thanks to the Network and Distributed System class for their help and for making this year an experience to remember!

Sarah Alzahrani

University of Dublin, Trinity College

28th August 2015

Abstract

In the past few years, there has been a considerable amount of growth in semantic web adoption. A large amount of the information available on the web has been modelled using various types of data model, each having a different form of standardization, format and rules governing the data. Thus, users are increasingly choosing to uplift data into Resource Description Framework (RDF) format, which is a standard model for data interchange on the web. However, because data schemas and ontologies that are used to describe data in RDF may change frequently over time, the need for an approach to represent the mapping statements in an analyzable form has arisen.

Most current research effort in this area is focused on demonstrating generic solutions for mapping, or translating data that exists in different formats from the standard data model. However, not much attention has been given to the representation of the mapping statements in the previous/current solutions. In this study, focus is placed on achieving the mapping process in an autonomic fashion. In other words, representing query-based mapping (XQuery queries) using RDF (Resource Description Framework), which appears (after significant investigation) to be a novel idea in the area. Therefore, this representation will enable the manipulation and analysis of mapping statements.

Two techniques are proposed to perform the mapping from XQuery to RDF, which are *manual* and *automatic* mapping approaches. The two approaches are based on one solution path that begins with the extraction of XML documents using XQuery to uplift data into the standard RDF data model. Then, the XQuery that is used for the mapping process itself is transformed into RDF to allow for further reasoning and analysis of the query. Finally, an evaluation approach was formulated in order to assess the performance of the two mapping approaches.

Table of Contents

List of Tables

List of Figures

Chapter 1 : Introduction

1.1 Motivation

Nowadays, with tremendous amounts of data involved in our everyday activities; organizations, institutions and enterprises (from banking to hospitals etc.) often choose to model their data independently in a specific way. However, various data models were introduced to handle large amounts of data, and with each model comes a different form of standardization, format and rules to govern the data.

With the growth in semantic web adoption, a challenge remains in extracting data from different data models into a standard model. Even though the available solutions that perform this mapping process could be used to achieve the required transformation, deficiencies still exist, as evolution and maintenance of the mapping statements themselves have not yet been addressed.

The most recent solutions in the area such as [1] and [2] have attempted to address the mapping problem by proposing and implementing languages and tools which enable transformation from different data formats to Resource Description Framework (RDF). RDF is defined as the main building block for semantic web, which provides a flexible standard data model to represent information in a way that facilitates interchanging and combining data from different data sets and applications [3]. However, not much attention has been given to the representation of mapping statements in the previous solutions.

The focus of this study is to propose a prototype to transform data from *XML* data models to a standard model. The main contribution of this work concerns processing the mapping statement itself and transforming it into machine readable form. The proposed solution (see Figure 1.1) begins with the extraction of XML documents using *XQuery* to uplift data into machine readable form, i.e. RDF. The XQuery that is used for the mapping process itself shall then be transformed into machine readable form to allow further reasoning and analysis of the query.

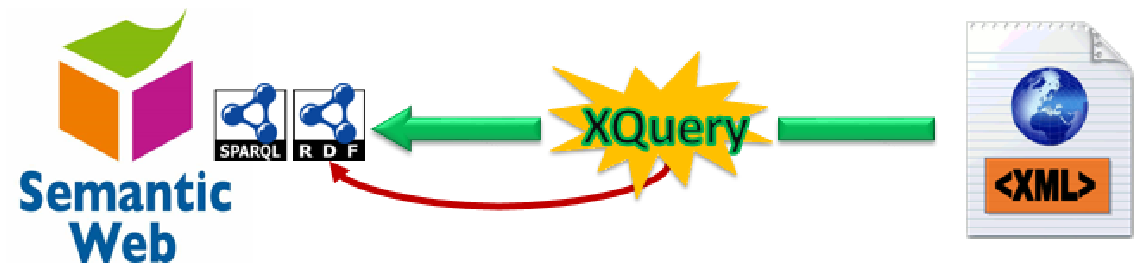


Figure 1.1 - Research goal.

1.2 Dissertation goal

Research question

The research question addressed in this thesis is:

"To what extent will representing query-based mapping (XQuery queries) using Resource Description Framework (RDF) allow for ongoing analysis and recomposition of mappings?"

Research objectives

The research question led to the following objectives:

- Investigate current state of the art methods in the area of data mappings and integration.
- Develop a prototype that allows users to translate query-based mappings into RDF based data models.
- Investigate the accuracy and mapping issues that arise in translating XQuery into an RDF based data model manually and automatically.
- Compare and contrast the automated translation from XQuery into RDF versus the manual translation.
- Analyze the produced output from converting XQuery into RDF by using SPARQL query language.
- Investigate the ability of XQuery to represent complex mapping.

1.3 Research methodology

This section provides an overview of the methodology followed during the implementation phase of this research.

In order to achieve the objectives of this research, an iterative methodology shall be adopted. Two iterations are required for achieving the implementation, whereby each iteration consists of three phases. During the first phase, several steps are applied in order to build an XML file which acts as base for the other phases. The main step in the first phase involves mapping XML data to the RDF data model based on a specific ontology using XQuery. In the second phase, the focus is converting the query-based mapping statement into machine readable form. During the second phase, two paths will be explored; the first path involves converting XQuery to machine readable form manually, based on a predefined specification, whereas the second path is aimed at automating the same process. The third phase in this research is the analyses and manipulating the output from the previous phase. As stated, the difference between the first and second iterations of this project is that the first iteration is performed in a manual fashion, whereas the second iteration will attempt to automate the same process.

1.4 Dissertation outline

Chapter 1 outlines the motivation behind this research, research question, research objective and the research methodology followed to conduct this study.

Chapter 2 gives an overview of the necessary background information in the area of semantic and linked data. The chapter discusses the basic building blocks for semantic mapping, i.e. RDF and some of its widely used serialization processes. Additionally, the chapter defines the differences between two essential concepts in relation to this research which are: *uplifting* and *interlinking*.

Chapter 3 is a literature review of current, state of the art research areas which are relevant to this research. The first part of this chapter investigates the current ongoing research in the area of mapping from specific or different formats to standard data model. Additionally, two query based approaches that influenced this research are discussed.

Chapter 4 illustrates a variety of design approaches which are considered for possible solutions. This design chapter defines the required functional and non-functional requirements. Then, the architecture of the project is presented with detailed illustrations of all components involved. Finally, all the design challenges faced during the project are highlighted along with the alternative design solutions taken.

Chapter 5 discusses the steps taken during the implementation phase of this research. The implementation chapter begins by defining programming languages and technologies used in implementing this solution. The actual implementation method is then presented, which involves three distinct stages.

Chapter 6 discusses the main contribution of this project and conclusions, as well as recommendations for future work in the field.

Chapter 2 : Background

2.1 Introduction

This chapter provides an insight into the background of the semantic web and linked data. The second part of this chapter introduces RDF, the main building block of semantic web [4]. Furthermore, several RDF serialization formats are introduced which are currently in use, and the RDF standard query language itself. The third part of this chapter introduces the differences between data uplifting and data interlinking.

2.2 The semantic web and linked data

According to Berners-Lee, web 1.0 was the first generation of the web and he described it as the read only web [5]. In web 1.0 (which is usually referred to as the classic web), hyperlinks are used to connect the documents which eventually create a single global information space. On the other hand, linked data allow connections to be made between entities that are located in different data sources. As a result, these connections will link sources into a single global data space. However, web standards and common data models are essential in order to implement generic applications that operate over this global data space. This is the essence of linked data [6].

Data publishers and application developers who adopted linked data are contributing in building this global interconnected data space which is referred to as the web of data. The web of data interlinks various domains such as people data, companies, geographical locations, publications, scientific and government data. This contribution from various domains will create seamless links between the datasets [6]. From the user's perspective, the key benefit of linked data is the access to data that is being collected from distributed and heterogeneous data sources. This access may reach data from sources not explicitly specified by the user [7]. This web of data is often called the semantic web and the term represents the World Wide Web Consortium's (W3C's) vision of the web of linked data. Semantic web allows people, governmental organizations and businesses to build data storage on the web, create vocabularies and govern this data by writing appropriate rules. To achieve the previous activities and to empower linked data, semantic web technologies such as RDF, SPARQL, and web ontology language (OWL) are utilized [8].

Figure 2.1 shows the semantic web technology stack known as a “semantic web layer cake”. The stack is a popular illustration of the key technologies in the semantic web.

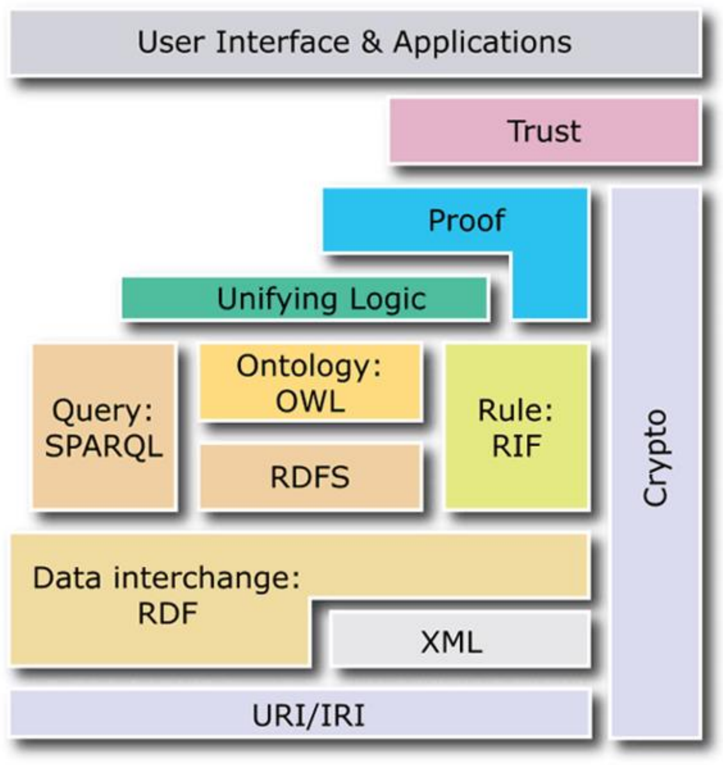


Figure 2.1 - Semantic Web Layer Cake (W3C).

2.3 RDF

Standard data modeling is required in order to connect the distributed data in the web which will specify the existence and the meaning of the links between the entities in this data. This is achieved by a standard graph-based data model known as RDF. Entities in the real world such as people, locations, products, books, diseases and even abstract concepts and the relations between them can be described using RDF in very simple way [6].

RDF as defined by W3C is “a standard model for data interchange on the Web. RDF has features that facilitate data merging even if the underlying schemas differ, and it specifically supports the evolution of schemas over time without requiring all the data consumers to be changed” [3].

RDF describes items in the form of triples which are sometimes called RDF statements. A triple is an atomic statement which consists of a subject, a predicate and an object. The subject refers to the item that the triple is describing. The predicate defines the characteristic of the subject that is being described in the statement. The object in the RDF statement is the actual value which could be literal or could refer to a subject of another statement [9]. The diagram in Figure 2.2 illustrates the concepts of the subject, predicates and object by a simple analogy.

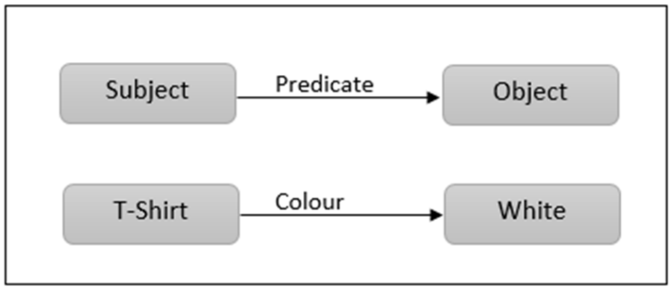


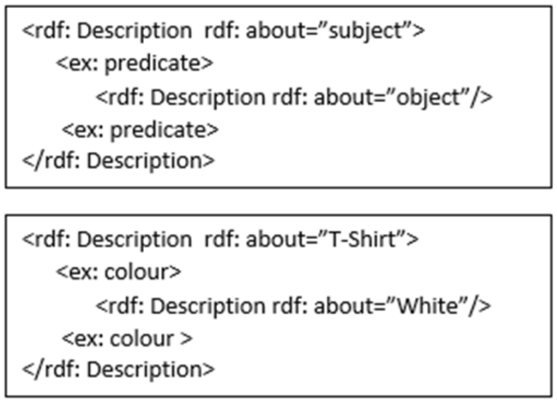
Figure 2.2 - RDF Triple in graphical form.

2.4 RDF serialization formats

Various RDF serialization formats are in use such as RDF/XML, Turtle, N-Triples and N-Quads. RDF/XML and Turtle are most popular RDF serialization formats widely used to express the triples.

2.4.1 RDF/XML

RDF/XML is the XML based RDF syntax which is considered as the first and the standard RDF serialization format [10]. RDF is not an extension for XML, but XML is one of the alternatives used for writing and exchanging RDF triples [11]. The following example in Figure 2.3 illustrates how an RDF triple can be represented using the RDF/XML syntax.



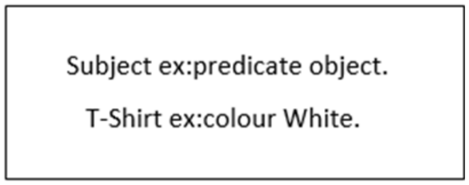
```
<rdf: Description  rdf: about="subject">
    <ex: predicate>
         <rdf: Description rdf: about="object"/>
     <ex: predicate>
</rdf: Description>
```

```
<rdf: Description  rdf: about="T-Shirt">
    <ex: colour>
         <rdf: Description rdf: about="White"/>
     <ex: colour >
</rdf: Description>
```

Figure 2.3 - RDF Triple in RDF/XML format.

2.4.2 Turtle

Turtle is a more readable and human-friendly syntax than RDF/XML for writing RDF statements [12]. It is not an XML syntax but was designed specifically for RDF. Triples in Turtle are represented in a very simple format which consists of a subject, predicate and object, written on a single line separated by a white space, with the full triple terminated by a period. Figure 2.4 shows an example of how a triple is represented using the Turtle syntax.



```
Subject ex:predicate object.
T-Shirt ex:colour White.
```

Figure 2.4 - RDF Triple in Turtle format.

2.5 RDF query language (SPARQL)

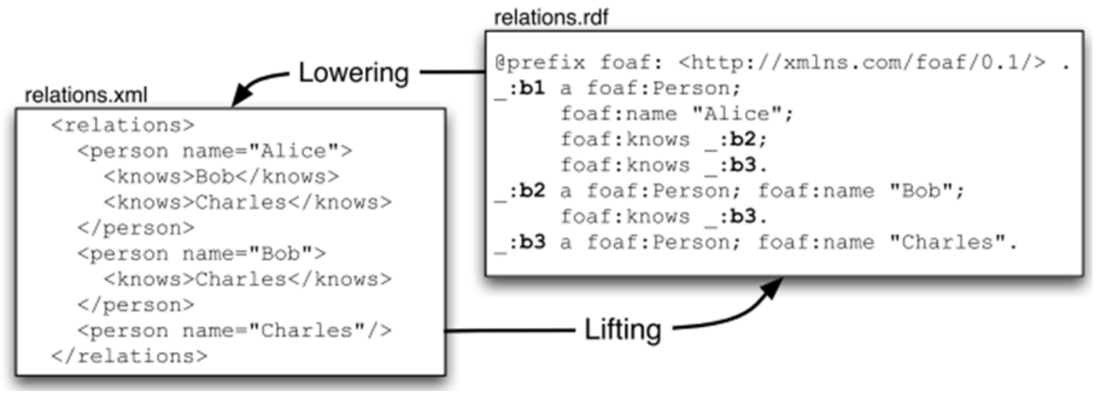
For tasks such as creating links between RDF triples and traversing through them, a structured query language is used which is referred to as SPARQL. SPARQL is not only a standard query language dedicated for querying RDF graphs but is also enables data manipulation over RDF triples [13]. In this research, SPARQL queries were written in order to evaluate the RDF files. Furthermore, SPARQL queried could be used to apply several types of operations such as insertion, deletion and manipulation.

2.6 Interlinking versus uplifting

Linked data is enabled by mechanisms used in the semantic web in order to interconnect the available resources in the web by using structure links. Uniform Resource Identifiers (URIs) and RDF are the structure links being used, which provide further benefits beyond the linking process such as making the discovery of information easier, providing clear representation of data and knowledge, as well as facilitating data integration processes [14]. The interlinking process is applied between the RDF datasets in order to connect them. However, this needs to be distinguished from the uplifting or lifting process that is addressed in this dissertation topic.

In the semantic web service community, software that uses RDF for modeling its data needs to exchange information with an XML based web service. As a result, a transformation is needed between RDF and the XML message. The semantic annotation for the Web Service Definition Language (WSDL) working group introduced the concepts of lifting and lowering [15], [16]. When the data is in RDF format, it is on higher level of abstraction than XML data. Because of this, the transformation from a lower level of abstraction to RDF is referred to as lifting. Conversely, lowering is used to refer to the opposite direction, which is the transformation from a higher level of abstraction to a lower level [17].

The lifting or uplifting process focuses precisely on translating from syntax to data representation. For example, translating data from an XML syntax to any RDF serialization format. Lowering, on the other hand, refers to the reverse process whereby data representation is translated to data syntax. An example of this could be mapping an RDF document to the equivalent XML file [18]. The diagram in Figure 2.5 shows an example illustrating the two concepts.



```
<relations>
  <person name="Alice">
    <knows>Bob</knows>
    <knows>Charles</knows>
  </person>
  <person name="Bob">
    <knows>Charles</knows>
  </person>
  <person name="Charles"/>
</relations>
```

```
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
_:b1 a foaf:Person;
     foaf:name "Alice";
     foaf:knows _:b2;
     foaf:knows _:b3.
_:b2 a foaf:Person; foaf:name "Bob";
     foaf:knows _:b3.
_:b3 a foaf:Person; foaf:name "Charles".
```

Figure 2.5 - From XML to RDF and the reverse direction [15].

2.7 Summary

In this chapter, the background of linked data, the semantic web and its building blocks were reviewed. Furthermore, the main concepts and different formats used in this research have been illustrated. This chapter therefore serves as an introduction to the state of the art chapter which follows.

Chapter 3 : State of the Art

3.1 Introduction

A vast amount of research has been undertaken in relation to the translation between different data formats and RDF in the field of semantic web. For the purpose of this research, the focus is placed on ongoing research attempting to demonstrate solutions for mapping or translating data which is different to the standard data model. Current research in the area such as [19], [1] and [17] are trying to build a generic and unified language for mapping purposes in the semantic web area. Limitations of the current mapping methods have been argued and the need for efficient mapping processes is increasing.

The main objective of this research is achieving the mapping process in an autonomic fashion. In other words, representing the query based mapping in an RDF data model, which appears (after significant investigation) to be a novel idea in the area. Other approaches are discussed in further detail later in this chapter such as XQueryX [20], have influenced this study in the context of translating XQuery to another format. However, XQueryX focuses on defining an XML representation for XQuery queries, whereas this research focuses on defining a way to represent XQuery in a machine readable form. Therefore, further analysis and manipulation could be applied over the queries.

This chapter aims to present a current summary of the current progress in research in the field of mapping in semantic web. The emphasis is placed on the approaches and techniques adopted in key studies in order to perform and represent the mapping process at present. An investigation of the current query based approaches used for mapping purposes is also presented.

3.2 Mapping formats (representations)

Several tools and languages exist to perform mapping from various formats and serializations to RDF. These tools allow translations from the most known and used formats such as relational databases, XML, CSV and spreadsheets [21], [22],[15], [23],[24]. However, according to Dimou et al., at the time that they proposed their solution which called (RML) RDF Mapping Language, a vast number of existing solutions performed mapping from XML to RDF. They claimed that there was no specific language designed for translating from XML to RDF without relying on XSL transformation (XSLT) and XPath for achieving the mapping. However, they argued that relying on XSLT and XPath adds unnecessary complication to the mapping process as these tools deal with the files as XML documents rather than RDF triples. Therefore, they proposed RML which is an extensible generic language [19]. RML is discussed in detail in the next section to investigate this language, how the mapping statements are represented and the ability of the language in representing complex mapping.

Representing complex mapping is a tedious task. However, most current research in the field of semantic web is aimed at addressing this task. In the world of ontology, finding matches between ontologies is not an easy task. Later in this chapter, the definition of correspondence patterns and the Expressive and Declarative Ontology Alignment Language (EDOAL) language is introduced in order to investigate their ability in representing complex correspondence patterns between ontologies.

In this study, XQuery is used to perform the mapping. However, one objective of this research is to investigate the ability of XQuery in representing complex mapping. For this reason, different mapping representations of the current existing solutions in the area are investigated.

3.2.1 RML

RML was originally aimed to overcome many limitations in the existing solutions at the time that it was introduces, as it provides a uniform, integrated form of mapping that translates data present in any format to RDF [25],[26]. Dimou et al. identify and discuss these limitations that motivated them to propose their solution. RML addressed the issues that appear in other existing solutions and prevented them from achieving a successful integration after the mapping. These limitations can be summarized as follows; firstly, most of the existing tools and languages apply mapping on one source at time rather than on multiple related sources together. Other limitations in existing solutions include *per-format mapping*, in which mapping is provided for each format independently. For example, a tool may only support mapping from XML to RDF. Mapping definitions in current solutions are not reusable and are tightly coupled to the implementation of a particular tool. Therefore, it is a complicated task to extract these mapping rules and reuse them in other applications that might handle the same domain concepts [19].

RML focuses on addressing the previous limitations during the mapping phase rather than later, in order to achieve a successful integration process and obtain a richer interlinking resource. Therefore, no further maintenance data cleaning is required after the mapping.

RML is an extension of the R2RML mapping language which is one of the W3C standards [21]. R2RML is a mapping language that focuses on expressing customized mappings from a relational database to an RDF dataset, while RML is a more generic language that extends the mapping to other formats [27]. In R2RML, a *triples map* is used to specify a rule for translating each row in a relational database into zero or more correspondence RDF triples. Since RML is a superset of R2RML, The mapping of data to an RDF data model in RML is based primarily on these triple maps which define the mapping rules that generate the RDF triples [17]. Triple maps consists of three main parts which are the logical source, subject map and zero or more predicate-object map. *Logical Source* refers to the input source which needs to be mapped to an RDF data model. The second part of the triples map is the *subject map* which indicate the used URI pattern in order to generate subjects for each corresponding RDF triple. The *predicate object* is the third part in triples maps. This part is divided further into two different parts: *predicate map* and *object map*. Predicate map identifies the predicate of the RDF triple, while object map specifies the object of that triple [19].

The main difference between RML and R2RML is the expected input, which is limited to relational databases in RML, whereas it can be of a different format in RML. Therefore, RML maintains the same mapping definitions as in R2RML but discards its database specific references. Furthermore, RML mapping definitions are formulated in the same syntax as R2RML [19].

To perform mapping using RML, the user needs to provide a well formatted data set as an input and a mapping definition that is specific to the mapping definition executed in order to produce the provided input in an RDF data model. Since RML deals with different formats, there are specific details which need to be specified before applying the mapping process. These are: *logical source, reference formulation, iterator, logical reference and reference object map*. The logical source is required to determine the input data source. Reference formulation needs to be specified in order to refer to the elements of the input data source. For example, if the input data is stored in an XML file, an XPath will be defined as the reference formulation. Also, an iterator needs to be determined for each potential input format. The iterator determines the iteration pattern over the provided input data set. The logical reference is used to determine a specific value within the input source [19].

The RML working group have introduced a semi-automatic method of producing RML mapping definitions more quickly [1]. They built an extension for Karma, which is an information integration tool. Using this tool, users are required to enter a semantic type for each data element in order to describe the relationships between the data elements. This tool provides users with suggested semantic types by reasoning of OWL ontologies, learning from the past action of the same user or using random fields. Also, Karma is able to visualize the entered semantic types, classes and relationships. Classes are represented as internal nodes and the semantic relationships between the classes are reflected as links that connect all of these classes. Semantic types in the graph connect data to classes [1].

In 2014, an RML working group introduced their novel approach of mapping data in both tabular (relational data bases and CSV) and hierarchal source (XML and JSON) into RDF using the RML language [1]. They noted that RML now supports an increased number of formats, whereas the existing implementation of RML supports only three types of data serializations at present, which are XML, JSON and CSV.

Evaluation of the RML language was provided in research [1], where the results were compared against those produced by a tool called *Datalift* [28], which was one of the pioneer solutions in converting from different data formats to RDF. The authors claimed that their solution that is based on RML is semantically richer and produces a better interlinked output. Finally, the authors stated in their future work that their further efforts will focus on data cleaning and enhancing the efficiency of RML processing by extending RML to support views on sources, built by queries.

3.2.2 EDOAL

One of the semantic web goals is to allow users and software agents to work in collaboration, sharing data and knowledge by expressing it in machine readable form. The semantic web community has proposed different technologies and standards in order to achieve this goal. Ontology is one which is referred to as "a formal specification of a shared conceptualization" [29]. Thus, an ontology may classify the concepts that can be used in a specific domain, the possible relationships between them, define constrains that govern these concepts, extend incomplete information and express negative information [30].

In the last few years, with the adoption of semantic web, the use of ontologies in various domains and applications has been increasing. However, when the meaning of two ontologies overlap, they can link together. This linking between ontologies is called *alignment*, which specifies a set of correspondences and each correspondence represents a link between a set of entities in two different ontologies [31].

Finding alignments between entities is not a simple task. However, according to Scharffe et al., at the time that they proposed their solution, the majority of the existing systems that find these alignments produced poor quality alignments. They argued that the other existing tools such as graphical user interfaces can help in designing the alignments manually, but they are not sufficient. Furthermore, they claimed that most ontology matching systems are limited to discovering only simple correspondence between entities and this is not sufficient to represent the relationships between entities. Therefore, they proposed a solution called *ontology correspondence patterns*. In this solution, they provide patterns of the most common correspondences amongst the alignments designers. Thus, building alignments can in most cases instantiate these patterns [31].

Correspondence patterns act as template which help identify more specific correspondences between entities. This process of identifying other correspondences is often called *pattern-based matching*, which usually starts from a collection of predefined correspondences. However, correspondence patterns need to use an expressive language In order to produce meaningful correspondences [32]. For this reason, EDOAL was designed.

EDOAL was introduced in order to provide alignment vocabulary to represent correspondences between entities of different ontologies [33]. Unlike the other available vocabulary and languages that represent the correspondences, EDOAL allows the capture of complex correspondences by applying the following features:

- ***Construction***: of entities from other entities which can be expressed using algebra operators. This feature allows the extension of incomplete ontologies.
- ***Restrictions***: can be applied to entities in order to limit their scope. This feature will result in providing better alignment between two sets of entities that belong to different ontologies.
- ***Transformation***: of property value is a feature used to align property values which have different encoding or units.

- ***Link keys***: are defined for expressing rules and conditions that specify when two entities are equivalent or not [33].

3.3 Query based approaches

Integrating data from different data models is not an obvious task. Different organizations and data publishers who would integrate their data might use various types of data formats, different structures and assign different conceptual models to their data which make the integration and exchange processes even worse. These problems often arise in the web, which are referred to data heterogeneity or data variety issues.

According to Bischof et al., there is a gap in the web of data between semantic web adoption and the availability of data that is in XML format. XML is considered as a frequently used data syntax for exchanging data and is rapidly applied. On the other hand, semantic web building block RDF is widely used and adopted by different parties in the academic and industrial sectors. Therefore, it is clear that switching between the two (XML syntax and RDF data model) is useful in many different cases. Although several tools and languages have been proposed and implemented over the years, the translation between these two worlds is not an easy task. The authors in [15] present a query based approach to address this issue called XSPARQL, which combines features from both XQuery and SPARQL.

Another query based approach that influenced this research is XQueryX which is a W3C recommendation. It is considered to be a way to define an XML syntax for XQuery [20]. The main benefits of transforming a query into an XML include the ability to query the XQuery query itself and manipulate it. In contrast to this approach, XQuery queries in this study are transformed into a machine readable form, i.e. RDF.

In the two following sections, XSPARQL and XQueryX along with some of their available implementations are investigated in further detail.

3.3.1 XSPARQL

One of the motivations behind XSPARQL is the lack of existing languages which transform between RDF and XML efficiently. The majority of solutions that have been proposed by the working groups who faced these issues in transferring between XML and RDF rely mostly on XSL Transformations (XSLT). XSLT as defined in research [34] is "a language for transforming XML documents into other XML documents ". However, XSLT presents issues when it is applied over RDF data that is in RDF/XML form. XSLT was designed to handle XML data with a simple hierarchal structure without any support for any data models such as RDF. RDF triples in XML format is quite different. Therefore, processors that manipulate RDF triples as pure XML document are required to consider the differences between the two representations [15].

Bischof et al. in [2], [35] have introduced various versions of XSPARQL (which is a scripting language that combines both SPARQL and XQuery in one query language). SPARQL and XQuery are languages used to perform processing over RDF and XML, respectively. SPARQL is a standard query language that is used to express queries over various kinds of data sources, stored or viewed as RDF [13], whereas XQuery [36] is a query language that is designed to run over data stored physically in XML files or viewed as XML via middleware. XSPARQL merges the two languages, which provides additional advantages. It allows developers to benefit from SPARQL features for retrieving RDF data and to construct RDF graphs using the Turtle syntax in addition to XQuery features for processing XML files [35]. XSPARQL is syntactically and semantically similar to XQuery, which is built on top of it by introducing a new type of for clauses called SparqlForClause. This clause is syntactically similar to the SPARQL select query [37].

XSPARQL can query RDF data sources and format data in RDF such as RDF/XML and Turtle. Also, it provides four different functions. The first is transformation between XML or JSON and RDF. The transformation processes can occur in both direction, for example: from XML to RDF (lifting) or from RDF to XML (lowering). Control flow to SPARQL is enabled by the additional advantages that XQuery added to the language. Other functionality includes scripting for web data integration and the ability to process relational data through RDB2RDF [2].

The language has several versions, with the latest release of it being XSPARQL 1.1 [2] which supports JSON documents as well XML to increase language usability. Figure 3.1Figure 3.1 illustrates the XSPARQL engine architecture which is written in Java and developed in a modular way. The architecture consists of the following primary components: an XSPARQL -rewriter and an XSPARQL -evaluator.

According to Bischof et al. [2], in their implementation the XSPARQL query, firstly it is necessary use a rewriter which is responsible for processing the XSPARQL query, parse it and rewrite the query in the form of an XQuery query. The output of the previous component (rewriter) will be the input of the evaluator which performs the next step. The purpose of the evaluator is to execute the query against the data and produce the answer by taking the XQuery as an input with embedded SPARQL queries. The evaluator will execute the queries and provide the collected results based on the specific query plan. Saxon and Jena-ARQ engines are used to execute the query plan for the XQuery query and SPARQL query, respectively.

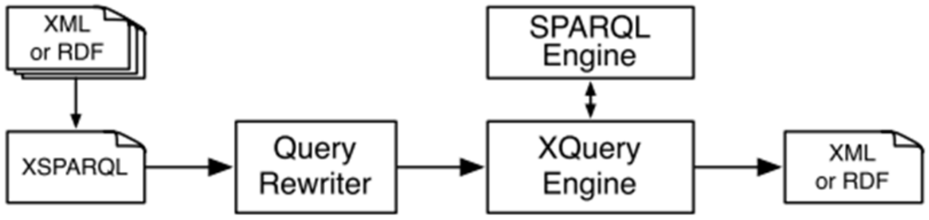


Figure 3.1 - XSPARQL Architecture [15].

Bischof et al. in [17], mentioned that a generic and unified language could be used for both querying and transforming data from one format to another. According to them, this could occur by extending XSPARQL. This extension will make it possible to transform data to several formats. However, it can be observed that recent, ongoing research in the area is attempting to build a successful generic and unified language for transformation purposes.

3.3.2 XQueryX

XQueryX works by simply mapping XQuery grammar into XML productions [20]. Therefore, the available tools and systems that are used with XML files can be used as well with XQuery in order to create, manipulate and interrupt queries. Also, this translation to XML will enable the queries themselves to be queried. To put it differently, running queries over another query is possible in these approaches. In order to achieve this, the query first needs to be transformed into the XQueryX form and then queries can be applied by running XQuery queries over it.

In [20], four possible environments were mentioned in which XQueryX might be useful. First, for reusing the parser in heterogeneous environments where one parse can generate XQueryX for a variety of systems used to execute a query. Second, XQueryX can be used to query other queries or transform them into new queries. For example, a query can be executed over a set of XQueryX queries in order to specify which query uses FLWOR expressions. FLOWR expression is an XQuery query with specific structure which consists of five parts (**F**or, **L**et, **W**here, **O**rder by, **R**eturn)[38].Third, it can be used in some environments such as XML oriented environments where representing XQuery in XML would be easier and more convenient to use, since XML tools might be available. The fourth environment is where XQuery needs to be embedded directly inside XML documents. In this case, representing XQuery in XML format will make the integration task easier.

A limited number of implementations are available for XQueryX. One of the available implementations that was used in this research to convert and XQuery query into an XML representation was [39]. Fundamentally, it could be described as an online W3C applet-based tool that enables users to translate XQuery expressions to XQueryX. As mentioned in [39], the parser performs its parsing process based on the XQuery grammar that is available in the page as JavaCC/JJTree files . Moreover, it produces an error if the syntax of the input is incorrect. The input and output from this tool will be examined in detail in the implementation and the evaluation chapters of this study.

Another available implementation for XQueryX is XQ2XQX, an XQuery to XQueryX transformation. It is a personal project that is controlled by the Numerical Algorithm Group (NAG) [40]. In contrast to the previous tool, this tool is quite different and depends precisely on using a set of XSLT stylesheets in order to perform the required transformation. In this case, users need to download stylesheets that are distributed in [41]. Furthermore, for applying the XSLT stylesheets, other requirements include downloading an XSLT processor and this must be taken into account (illustrated in the official distribution page).

To convert an XQuery to its XML representation using XQ2XQX, two stylesheets need to be applied on the Query. Firstly, an XML file must be created using one stylesheet, which will act as an input for the second stylesheet. Therefore, researchers and developers prefer to use the online applet based too.

3.4 Summary of state of the art research

This state of the art review has presented ongoing research in mapping from different formats into RDF data model format. More precisely, methods have been investigated for the mapping statements represented in current solutions. It has been found that not much not much consideration has been given to the representation of the mapping statements in the previous solutions. Their focus is placed on trying to build a generic and unified language for mapping purposes in the semantic web area.

A comparison between RML, XSPARQL and this project is presented in Table 3-1. These approaches are compared against each other based on how the mapping statements are represented, the supported input formats and the type of mapping. *XQR* in the table is the name given to the manual approach of this research. *XQX2RDF* is the proposed name for the automatic approach in this project.

Mapping statements in RML are called mapping definitions. In this mapping definition, specific details about the input source format need to be provided for each input. The current implementation of RML accepts inputs in XML, JSON and CSV formats and will convert them into RDF. The final released implementation of RML discussed in research [1] performs the mapping in semi-automatic fashion, requiring users to enter specific information in order to build the mapping definition. Thus, in order to use RML for the mapping process, an input data source and information regarding mapping definitions are required.

The mapping statement in an XSPARQL implementation is an XSPARQL query which consists of an XQuery query and nested SPARQL query. The current implementation of XSPARQL performs the transformation automatically; users only need to upload the input data source and select the output format.

In this research two mapping stages are required in order to uplift an XQuery query into RDF data model format. First, data needs to be uplifted from XML syntax to RDF by using XQuery as a query based mapping approach. Then, the XQuery query itself shall be transformed into RDF. In the former, the mapping statement is an XQuery query and the supported format shall be an XML data source. Furthermore, the mapping type is manual because an XQuery query needs to be written to execute the mapping. In the latter mapping (which is the main objective of this research), an XQuery query will be uplifted either manually or automatically. The expected input for these two approaches is an XQuery query.

Table 3-1 Comparison of approaches.

Approaches	**Mapping representations**	**Supported format**	**Type of mapping**
RML	Mapping definition (logical source, reference formulation, iterator, logical reference and reference object map)	XML JSON CSV	Semi-automatic
XSPARQL	XSPARQL query	RDF XML	Automatic
XML to RDF	XQuery query	XML	Manual
XQR	Based on vocabulary specification	XQuery	Manual
XQX2RDF	Based on XQueryX	XQuery	Automatic

Chapter 4 : Design

4.1 Introduction

This chapter presents the solution design of the project which contributes to the final implementation. The first part of this chapter defines the functional and non-functional requirements. The architecture of the project is presented with detailed illustrations of all components involved. Finally, all the design challenges faced during the project are stated along with the alternative design solutions taken.

4.2 Requirements

The following two sections present both functional and non-functional requirements for this project. The main goal of this dissertation is to represent query-based mapping (e.g. XQuery queries) in a format that allows easy analysis and recomposition, i.e. RDF. The first step towards achieving this goal is choosing a use case domain. Publication data in Bibtex format was collected and is processed during different stages of this research. This step acts as basic block for all the other functional requirements that are listed below. The first four functional requirements contribute to building a basic infrastructure for the project.

4.2.1 Functional requirements

1- Collect publication data in Bibtex format.

2- Convert publication data from Bibtex format into a processable format (XML).

3- Identify ontology that has classes and properties suitable for describing publications entries in RDF data model.

4- Perform the mapping between the XML schema of the publication data and the selected ontology.

5- Uplift the XML file into machine readable form (RDF) by using a query based mapping approach (XQuery).

6- Validate the RDF output with predefined RDF schema using an RDF validator.
7- Convert XQuery into a form that enables automatic analysis and manipulation (RDF).
8- Query the RDF version of XQuery by using XSPARQL.

4.2.2 Non-functional requirement

1- Programming language: users who will apply the mapping need to have background knowledge of the following semantic web technologies: RDF in Turtle syntax and SPARQL query language. Also, familiarity with XML and XQuery is required.
2- Effort involved in performing the mapping process should be minimized.
3- Documentation: the specification that is used in carrying out the mapping needs to be provided and become available as an online version.
4- Dependency on other parties should be taking under consideration.

4.3 Architecture

This section presents the architecture of the system which is carried out in two levels of abstraction. At an early stage during the design phase, the workflow was determined in a high level of abstraction that would lead to achieving all the requirements of this research. Detailed architecture was then presented to illustrate all components involved in this research, as well as the input and output of each component and the flow of data.

4.1.1 Architecture (high level of abstraction view)

Architecture of the project prototype in a high level of abstraction (Figure 4.1) is divided into three layers, where the output of each layer is an input of the layer below it.

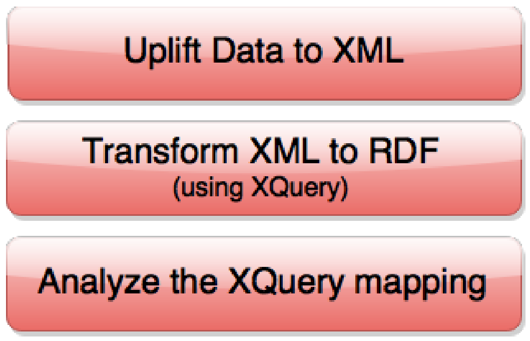


Figure 4.1 - Architecture (high level of abstraction).

- **First Layer** (Uplift data to XML)

After specifying the use case domain, data in Bibtex format was collected. However, this data needs to be mapped into a processable form. This layer reflects the first phase that is undertaken during the implementation phase which uplift data into XML. This layer can be eliminated if suitable data in XML syntax is available.

- **Second Layer** (Transform XML to RDF)

The goal of this project, as mentioned in the introduction, is to represent query-based mapping (e.g. XQuery queries) in a format that allows further manipulation and analysis. In order to achieve this goal, data must be mapped in machine readable form using XQuery as a query based approach for applying mapping.

- **Third Layer** (Analyze the XQuery mapping)

This layer focuses on representing XQuery queries in a form that is machine readable and processable. Two paths shall be taken and different design decisions are to be made during this phase, described in greater detail in the next section.

4.1.2 Architecture components

Dependency in the proposed solution is high, whereby each layer receives an input from the layer above it. The data flow in this prototype is illustrated using a pipelines pattern. The following two diagrams present the components of the solution in further detail along with the sequence of data flow (Figure 4.2).

Figure 4.2 shows all the required steps to convert publication data in Bibtex syntax into RDF, which is equivalent to layers 1 and 2 in the previous section. Figure 4.3 reflects the detailed processes of layer 3 which aims to convert XQuery queries into RDF.

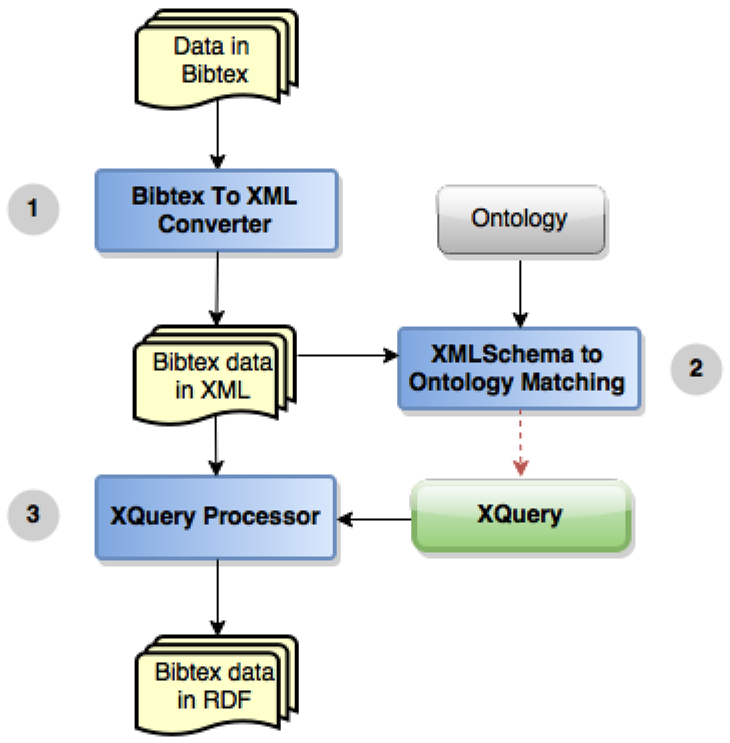


Figure 4.2 - Architecture (low level of abstraction -1).

- The first step in the solution, as illustrated above, is converting Bibtex to XML by using a Bibtex to XML converter.
- Next, identify a suitable publication ontology.
- Match XML Schema and the chosen ontology manually.

- Write an XQuery query by using the XQuery processor to translate data represented in XML into its RDF form. The mapping in this step should be done based on the previous step.
- The XQuery query that is written to perform the mapping is used as an input for the next two processes (see Figure 4.3).

Converting XQuery into RDF is what must be achieved. Two paths were proposed for the translation process, which are converting it manually or automatically (see Figure 10).

- The first path converts the XQuery to RDF manually based on a predefined specification.
- The second path is about automating the translation process, where the XQuery query is translated into XQueryX [20]. Then, XQueryX is translated into RDF by using the XQuery to RDF converter.
- Finally in this phase, SPARQL queries are run over the output from the two paths. The output is XQuery query represented in RDF data model.

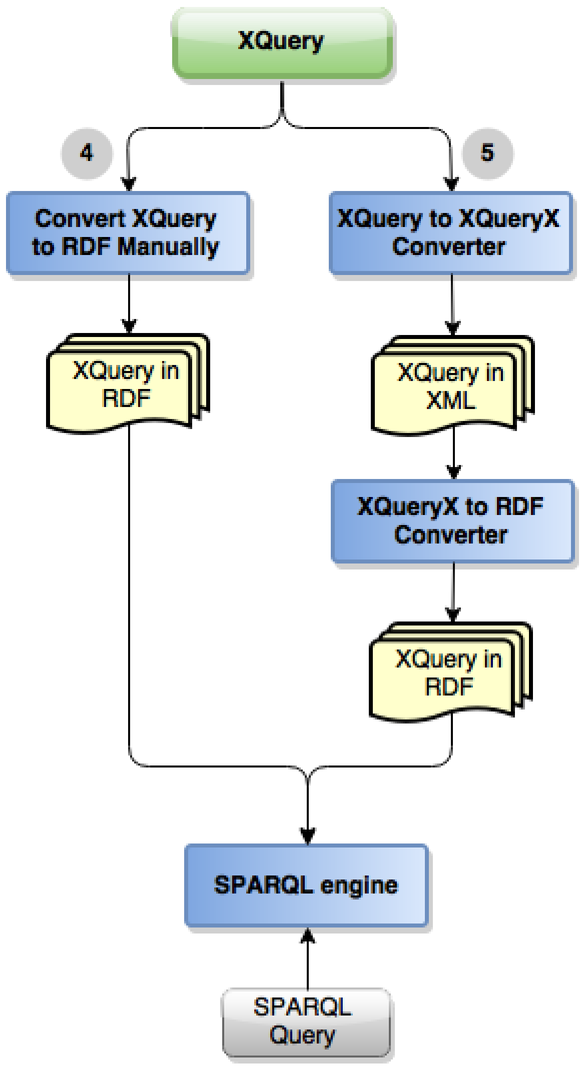


Figure 4.3 - Architecture (low level of abstraction – 2)

4.4 Design challenges

The research question which is the subject of this dissertation (outlined in Chapter 1) is "To what extent will representing query-based mapping (XQuery queries) using RDF (Resource Description Framework) allow for ongoing analysis and recomposition of mappings?". This question led to another important question and other challenges which are discussed in the following points.

1- How to represent XQuery in an RDF data model?

This question can be considered to be the primary challenge in this research. However, following a deep review of the semantic web area in Chapter 3, it is evident no one has proposed an adequate solution for the problem to be tackled in this research. Therefore, different approaches are proposed to build basic solutions for this problem and answer the research question. Details of how this challenge is tackled are discussed in the next section of this chapter.

2- Into which RDF serialization format should data be mapped?

In this project, two types of mapping into RDF is required (see Figure 4.1), which are XML to RDF mapping and XQuery to RDF mapping. As mentioned in Chapter 2, the RDF data model has different serialization formats that can be applied over any kind of data in order to represent them in the form of triples. The most widely used formats in RDF are XML format or Turtle format. The next section discusses which format is more suitable for the purpose of this research.

4.5 Design decisions

The previous section outlined the design challenges encountered during this project. This section presents the design decisions addressing these challenges.

1- How to represent XQuery in an RDF data model?

The first step required in this project in order to represent XQuery in an RDF data model will be to apply the mapping manually based on a proposed initial design undertaken by a postgraduate student in the Knowledge and Data Engineering group. The mapping type in this approach is manual mapping, where XQuery queries are translated into RDF data model format based on a predefined specification. This step contributes to identifying potential issues that might arise later in the project. It also forms the main motivation behind moving to the automatic solution.

The manual approach will not be feasible, especially when the mapping is more complex. However, the second answer to this question moves towards automating the mapping process, whereby XQuery queries are represented in a form that enables the use of an available mapping tool to convert them automatically into RDF.

The next proposal is mapping XQuery queries to RDF by writing an XQuery to perform the mapping automatically. This solution shall not be considered for various reasons which will be illustrated in the next chapter.

Another proposed solution shall involve building a parser which will perform the mapping based on XQuery and RDF grammar. This solution shall not be explored due to the time constraints of this research, but could be proposed as future work.

2- Into which RDF serialization format should data be mapped?

In this project there are two different stages where mapping to RDF shall be applied. In the first stage, the mapping will be between XML and RDF formats by using XQuery query as a query based mapping approach. An XQuery that perform the mapping will produce an RDF data model. However, uplifting data into RDF/XML format will absolutely result in writing a lengthy XQuery because of the XML hierarchies. For example, writing an XML opening and ending clause is required in order to build an XML/RDF data model. On the other hand, the mapping from XML into a Turtle representation will result in writing a shorter XQuery query than the RDF/XML query. The reason behind this is that Turtle format is more textual, where concatenation, white spaces and new lines could be used to construct the mapping result. As a result, converting XML to RDF in Turtle format will simplify the mapping of an XQuery query into RDF.

The second mapping stage will be the mapping between XQuery and RDF either manually or automatically. In the manual approach, representing an XQuery query in Turtle format is more suitable for the reason that Turtle is more human-readable. Representing the RDF result in RDF/XML or Turtle in the automatic approach will not affect any part of the mapping process.

4.6 Summary

This chapter discussed both functional and non-functional requirements that led to the final implementation. Next, the architecture of the solution and all involved components are introduced; firstly in a high level of abstraction and then in a lower level where details of the data flow and both inputs and outputs for each component are illustrated. Last but not least, all the design challenges and solutions to tackle them shall be described.

Chapter 5 : Implementation

5.1 Introduction

This chapter details the work that was undertaken during the implementation phase. Furthermore, it defines how the requirements of this project are implemented. The implementation phase is entirely based on the proposed layered architecture discussed in the previous chapter. The implementation chapter begins by defining programming languages and technologies that are used in implementing this solution. Then the actual implementation is discussed, which occurs over three different stages.

5.2 Programming languages and technologies

Various tools, query programming languages and semantic web technologies were used during the implementation phase of this research. The relevant query languages and semantic web technologies are defined below, along with the main reasons for selecting them for this project. The other used tools are discussed in detail later in this chapter.

- **XML**: was designed as way to describe data in a human readable format. It also plays an increasingly important role in representing data and facilitating the exchange process in the World Wide Web and elsewhere [42]. Data in this project is uplifted from XML representation to an RDF data model.
- **RDF**: is the main building block of the semantic web. It is a standard data model that supports data interchange and interlinking on the web [3]. In this research, XML files and XQuery queries are uplifted to RDF. The uplifting process is defined in detail in section 2.5.
- **XQuery**: is generally defined as an XML query language. However, the main objective of this research is to identify a way to represent XQuery queries in a machine readable form to allow further reasoning and analysis of the query. To achieve this, XQuery query is used during the mapping between XML and RDF data model. Then, it is converted into an RDF data model which will enable the analysis of the query.

- **XQueryX**: is an XML representation of an XQuery. After mapping an XML document into the correspondence RDF document by using XQuery as mapping approach, an implementation of XQueryX is used as an intermediate step in this project to represent the XQuery query used in the mapping process in an XML form.
- **SPARQL**: at the end of the implementation phase after the XQuery is represented in an RDF data model, SPARQL is the language used to query the RDF document for analysis and reasoning purposes.
- **Bibtex**: is a data format which is used to describe lists of references and is primarily associated with *LaTeX* documents [43]. The first step in this project was to choose a case domain and collect data suitable for the purpose of the project. Publication data in Bibtex format was converted into an XML representation in the early stages of this project.

5.3 Implementation

This section describes how the proposed approaches in section 4.5 were implemented in order to reach the desired goals. The approaches discussed in section 4.5 outline various proposed solutions that focus on how to represent XQuery in RDF data model. Two of the proposed solutions were considered at this stage of the project and discussed in detail within this section. The challenges which make the other proposed solutions difficult to achieve are also discussed in further detail in this section. The implementation phase is described below based on the high level view of the proposed architecture in section 4.1.1, which led to dividing the project into three different layers, where each layer represents one step towards achieving the goal. Tools used during these stages as well as inputs and outputs from these tools are also discussed in details.

5.4 Uplift of data to XML

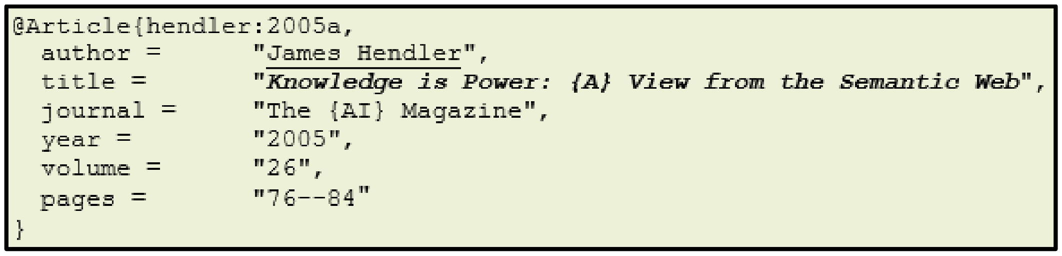
The first stage of this project is building the basic infrastructure, where the focus is placed on choosing a use case domain and collecting suitable data for the purpose of the research. Then, the collected data will be converted into machine processable format, which is XML.

As the name implies, in this stage data is uplifted from Bibtex format into XML. The XML output will be translated in the second stage into an RDF document. Even though there are already at least six available tools which translate from Bibtex to RDF format immediately [44], an intermediate step (which is the conversion to XML) is needed for the purpose of this research. In this stage, data can may be in any alternative format that represents data in a structured way, such as data in JSON format, data stored in a relational database or CSV format. Furthermore, this step can be eliminated if suitable data of the chosen domain is available in XML format.

The process of mapping Bibtex format to XML is initiated by firstly collecting publication data in Bibtex format from “The Collection of Computer Science Bibliographies”. This is a collection of bibliographies of scientific literature in computer science which is collected from various sources [45]. Furthermore, it contains three million references (i.e. journals, articles, technical reports and conference papers) in Bibtex format.

Figure 5.1 shows an example of Bibtex formatted data which represents one Bibtex entry. Each Bibtex entry starts by declaring the type of entry, which is the word appearing after “@”, followed by a key that is used to cite the entry. In Bibtex Description Format, there are different standard entries which cover most types of publications such as books, articles, theses and proceedings. The entry type in this example is “article” and the citation key is “handler: 2005a”. After this, lists of key-value pairs are presented. The entry type determines if the key-value pairs are mandatory or optional [43] [46].

In this project, only Bibtex entries of type *article* were collected. The reason for collecting only article entries is to facilitate the mapping process at this stage and to focus specifically on the main objective of the research which is representing the XQuery query used in the mapping process in RDF. However, all article entries consist of the same key-value pairs which are *author, title, journal, year, volume and pages*. Finally, the collected entries are saved as a file ending with *.bib* as a file extension, i.e. *List.bib*. The List. Bib file contains the list of Bibtex entries of type article which have the same structure of the presented example in Figure 5.1.

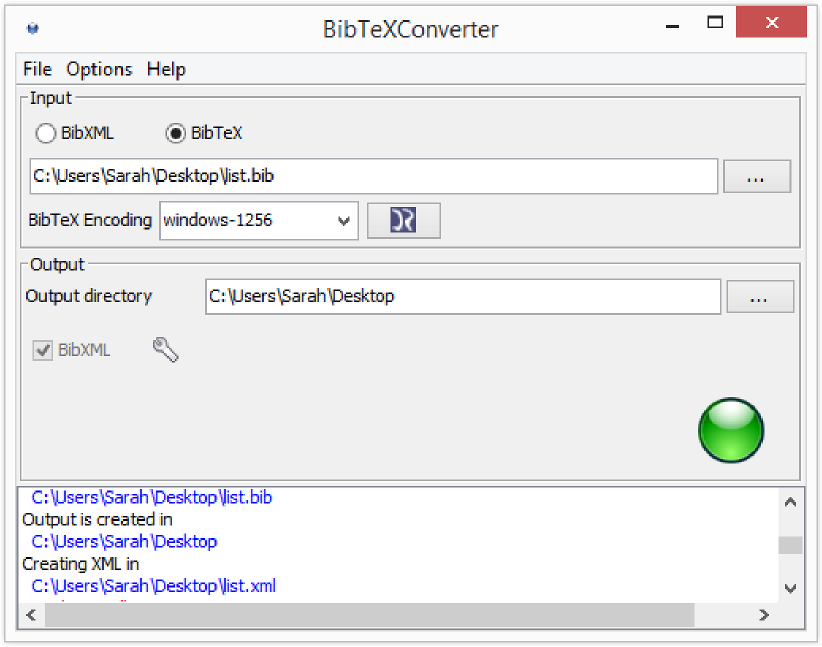


```
@Article{hendler:2005a,
  author =        "James Hendler",
  title =         "Knowledge is Power: {A} View from the Semantic Web",
  journal =       "The {AI} Magazine",
  year =          "2005",
  volume =        "26",
  pages =         "76--84"
}
```

Figure 5.1 - Example of Bibtex formatted data.

After collecting the data, a Bibtex to XML converter is used in order to map Bibtex entries into an XML representation. Nowadays, various tools exist to translate between the two formats. In this project, a tool called "BibTeXML" was used for the mapping process. BibTeXML is "a bibliography schema for XML that expresses the content model of Bibtex". BibTeXML converts Bibtex to XML and derives all its outputs by applying XSLT stylesheets to intermediary XML data [47],[48].

Figure 5.2 shows the main interface of BibTeXML, where the user can choose the input format and the file to be transformed. Bibtex is the input format and the file called "List.bib" is the file being converted, which holds one Bibtex entry (example in Figure 5.1). Finally, the output "List.xml" which contains the entry in XML format will be created in the selected directory. Figure 5.3 shows the XML representation of the same example in Figure 5.1. This XML file will be used in the next section in order to perform the uplifting process from XML to RDF.



Input: ( ) BibXML (●) BibTeX

C:\Users\Sarah\Desktop\list.bib

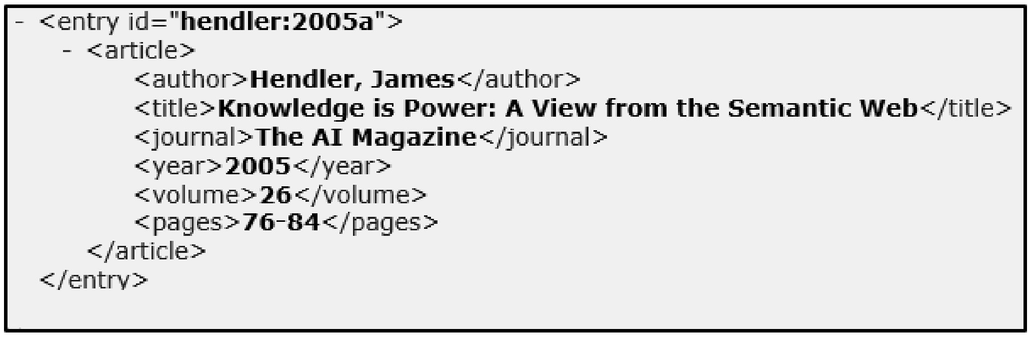
BibTeX Encoding windows-1256

Output directory C:\Users\Sarah\Desktop

✓ BibXML

```
 C:\Users\Sarah\Desktop\list.bib
Output is created in
 C:\Users\Sarah\Desktop
Creating XML in
 C:\Users\Sarah\Desktop\list.xml
```

Figure 5.2 - The main interface of BibTeXML converter.



```
- <entry id="hendler:2005a">
   - <article>
        <author>Hendler, James</author>
        <title>Knowledge is Power: A View from the Semantic Web</title>
        <journal>The AI Magazine</journal>
        <year>2005</year>
        <volume>26</volume>
        <pages>76-84</pages>
     </article>
  </entry>
```

Figure 5.3 - XML representation of the Bibtex formatted data.

5.5 Transform XML to RDF (using XQuery)

The title of this dissertation is "Autonomic Uplift of Information". The word "uplift" or sometimes simply referred to as "lift" is the process of converting data from a lower level of abstraction into RDF. Data in RDF is considered to be of a higher level of abstraction than data in XML format [17]. In this stage of the project, List.xml from the previous example is translated into RDF by using XQuery as a query based approach of applying mapping. Before performing the actual mapping, a suitable ontology for describing publication data in RDF should be selected first. A manual mapping between the XML schema and the selected ontology is then applied.

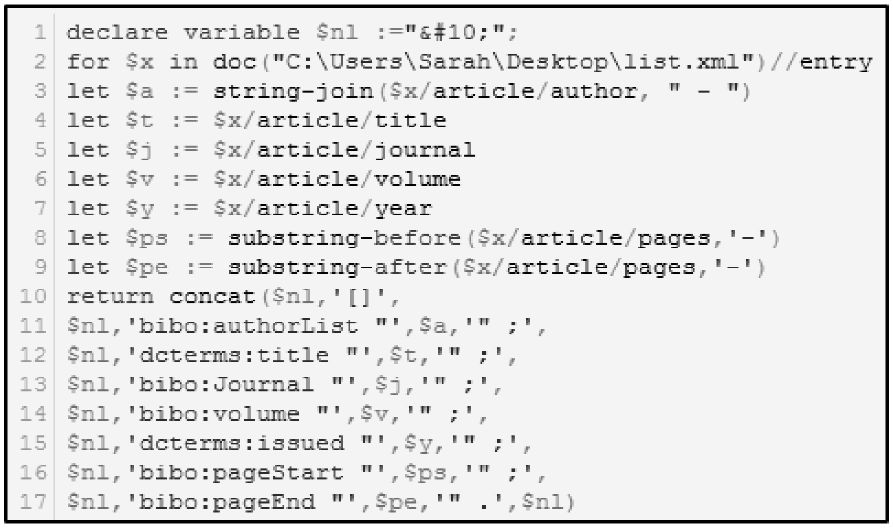
"The Bibliographic Ontology" (http://purl.org/ontology/bibo/) is the ontology used to describe the publication data in this project. It describes bibliographic data in the semantic web in RDF format. It can be used as a citation ontology, document classification ontology or for describing publications in RDF [49]. As shown in Figure 5.3, the article is described in XML syntax by using the following XML elements: author, title, journal, volume and pages. Based on these elements, the suitable properties from the bibliographic ontology are selected to describe them in RDF. Table 5-1 shows the correspondence properties from the selected ontology for each XML element.

Table 5-1 Mapping between XML and bibliograpgic ontology

XML element	Propriety
Author	bibo:authorList
Title	dcterms:title
Journal	bibo:Journal
Volume	bibo:volume
Year	dcterms:issued
Pages	bibo:pageStart bibo:pageEnd

After the manual mapping between XML elements and the ontology, an XQuery query is written to translate the XML file “List.xml” to RDF file “List.rdf”. At this stage, an RDF serialization format needs to be selected before writing the query. The most widely used formats are RDF/XML and Turtle, where the output of both represents the same data without any changes, although the representation is different. Turtle is a more human-readable format than RDF/XML. At this stage, XQuery, that converts the XML file to both formats, is provided in order to show the difference between them.

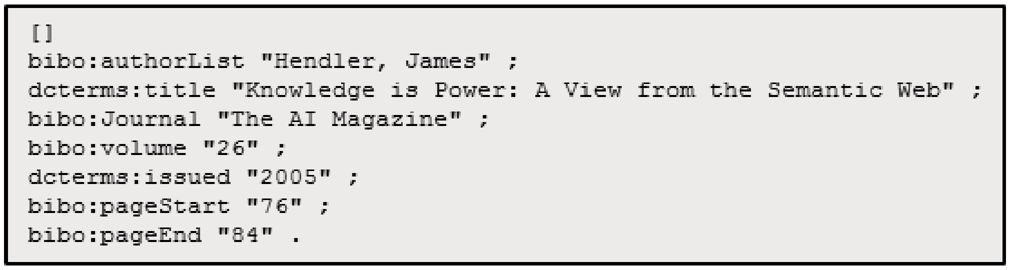
In order to run an XQuery query over XML documents, one of the available XQuery processors should be used. *BaseX* was chosen in this project for the reason that it is easy to use and offered a more user-friendly Graphical User Interface (GUI) than other available processors. Furthermore, it is acts as an XML Database engine and XPath/XQuery 3.1 processor at the same time. A basic XQuery FLOWR expression was written to iterate through the entries in the XML document in sequence. The FLOWR expression consists of five parts which are *for, let, where, order, return*. “For” identifies which entries are to be iterated through, whereas “let” is used to create temporary variables that will be used later in the return clause. “Where” is an optional part of the FLOWR expression that specifies conditions over the returned result. “Order” is used to change the order of the returned result. “Return” is the only required part in the FLOWR expression which defines the structure of the returned result [38]. Figure 5.4 shows a basic XQuery FLOWR expression that processes the “List.xml” document and converts it into Turtle format.



```
 1  declare variable $nl :="&#10;";
 2  for $x in doc("C:\Users\Sarah\Desktop\list.xml")//entry
 3  let $a := string-join($x/article/author, " - ")
 4  let $t := $x/article/title
 5  let $j := $x/article/journal
 6  let $v := $x/article/volume
 7  let $y := $x/article/year
 8  let $ps := substring-before($x/article/pages,'-')
 9  let $pe := substring-after($x/article/pages,'-')
10  return concat($nl,'[]',
11  $nl,'bibo:authorList "',$a,'" ;',
12  $nl,'dcterms:title "',$t,'" ;',
13  $nl,'bibo:Journal "',$j,'" ;',
14  $nl,'bibo:volume "',$v,'" ;',
15  $nl,'dcterms:issued "',$y,'" ;',
16  $nl,'bibo:pageStart "',$ps,'" ;',
17  $nl,'bibo:pageEnd "',$pe,'" .',$nl)
```

Figure 5.4 - XQuery FLOWR expression to convert an XML file into RDF (Turtle).

In Figure 5.4, line 1 contains the declared variable used to create a new line between the predicates. Furthermore, it is used only to represent the triples clearly. Thus, it is not essential for constructing the triple and it can be removed. Line 3 in the query defines the first let clause which refers to the authors of the article. The *string-join* function is used to apply the first mapping snippet which maps between the author element in XML and the authorList property in RDF. One article in XML may have multiple author elements. Therefore, the string-join function combines the authors of one article and separates them by using "– ". Title, journal, volume and year elements in XML are the same in RDF and no mapping is needed between these elements and the used ontology. Substring-before and substring-after are the second and third mapping snippets which map between the pages element in XML and PageStart, PageEnd properties in the bibliographic ontology. The Turtle output from running XQuery FLOWR expression over "list.xml" is shown in Figure 5.5.



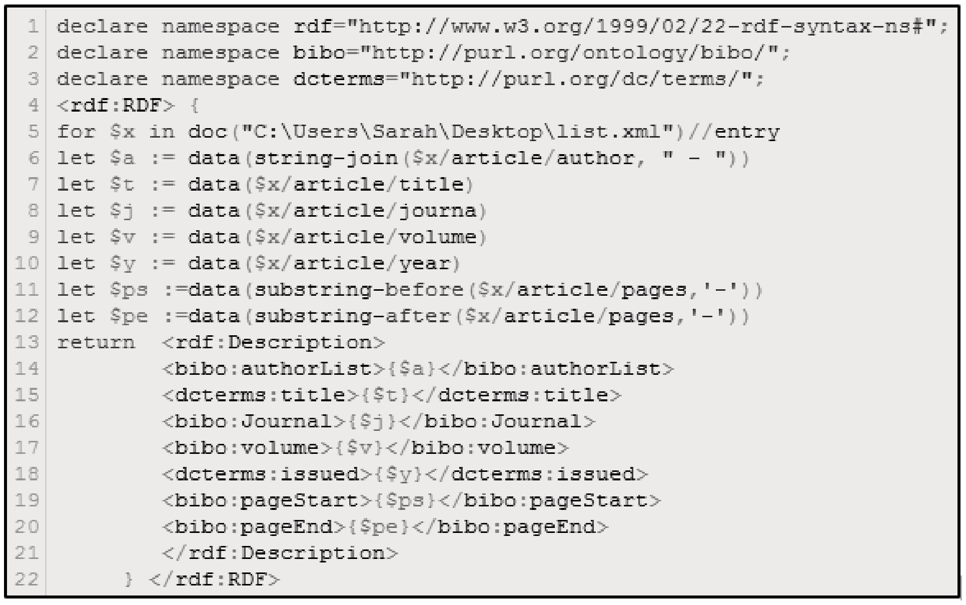
```
[]
bibo:authorList "Hendler, James" ;
dcterms:title "Knowledge is Power: A View from the Semantic Web" ;
bibo:Journal "The AI Magazine" ;
bibo:volume "26" ;
dcterms:issued "2005" ;
bibo:pageStart "76" ;
bibo:pageEnd "84" .
```

Figure 5.5 - The Bibtex formatted data in RDF data model (Turtle).

The Turtle document enabled an RDF graph to be written in textual form. However, an RDF graph contains multiple triples where each triple consists of a subject, predicate and object. A Turtle statement in its simplest format is made up of a subject, predicate and object written on a single line separated by a white space, with the full triple terminated by a full stop. By taking into account the chosen use case domain (which is publication data) where each type of publication has several properties, an alternative representation of Turtle statements is applied to represent the article and its properties. This alternative option is more flexible and clear in representing multiple predicates which belong to the same subject [12].

In this representation which is shown in Figure 5.5, a series of RDF triples are expressed. The subject of the triples in this example is a blank node. Blank nodes in Turtle represent subjects that are not assigned to IRI or literal. It can be represented explicitly with an identifier using this symbol ":_" followed by any name, or without names using another symbol “[ ]”. The subject in Figure 5.5 is referenced by seven predicates and objects, separated by a semicolon, “;”. Thus, the semicolon symbol is used to replicate the subject of triples that differ only in predicate and object [12].

Figure 5.6 shows an XQuery query that converts the same article entry into RDF/XML. The return clause in this query is more structured and organized than the Turtle query. The reason for this is that RDF/XML output written in XML and BaseX supports XML syntax as well as declaring namespaces. Figure 5.7 shows the same article output in RDF/XML after executing the XQuery in Figure 5.6.



```
declare namespace rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#";
declare namespace bibo="http://purl.org/ontology/bibo/";
declare namespace dcterms="http://purl.org/dc/terms/";
<rdf:RDF> {
for $x in doc("C:\Users\Sarah\Desktop\list.xml")//entry
let $a := data(string-join($x/article/author, " - "))
let $t := data($x/article/title)
let $j := data($x/article/journa)
let $v := data($x/article/volume)
let $y := data($x/article/year)
let $ps :=data(substring-before($x/article/pages,'-'))
let $pe :=data(substring-after($x/article/pages,'-'))
return  <rdf:Description>
        <bibo:authorList>{$a}</bibo:authorList>
        <dcterms:title>{$t}</dcterms:title>
        <bibo:Journal>{$j}</bibo:Journal>
        <bibo:volume>{$v}</bibo:volume>
        <dcterms:issued>{$y}</dcterms:issued>
        <bibo:pageStart>{$ps}</bibo:pageStart>
        <bibo:pageEnd>{$pe}</bibo:pageEnd>
        </rdf:Description>
     } </rdf:RDF>
```

Figure 5.6 - XQuery FLOWR expression to convert an XML file into RDF (RDF/XML).

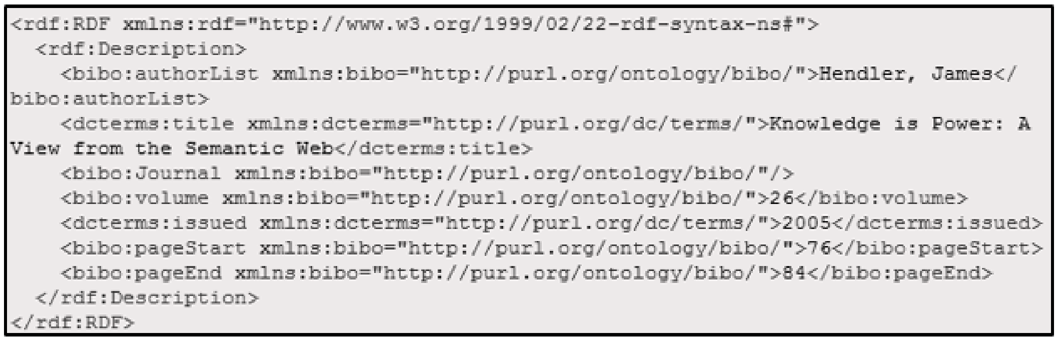

```
<rdf:RDF xmlns:rdf="http://www.w3.org/1999/02/22-rdf-syntax-ns#">
  <rdf:Description>
    <bibo:authorList xmlns:bibo="http://purl.org/ontology/bibo/">Hendler, James</
bibo:authorList>
    <dcterms:title xmlns:dcterms="http://purl.org/dc/terms/">Knowledge is Power: A
View from the Semantic Web</dcterms:title>
    <bibo:Journal xmlns:bibo="http://purl.org/ontology/bibo/"/>
    <bibo:volume xmlns:bibo="http://purl.org/ontology/bibo/">26</bibo:volume>
    <dcterms:issued xmlns:dcterms="http://purl.org/dc/terms/">2005</dcterms:issued>
    <bibo:pageStart xmlns:bibo="http://purl.org/ontology/bibo/">76</bibo:pageStart>
    <bibo:pageEnd xmlns:bibo="http://purl.org/ontology/bibo/">84</bibo:pageEnd>
  </rdf:Description>
</rdf:RDF>
```

Figure 5.7 - Bibtex formatted data in RDF data model (RDF/XML).

This RDF/XML example is presented in order to compare it with the Turtle example. Also, this comparison will clarify the reason behind choosing Turtle format to represent the mapping in this project over the RDF/XML format. It can be observed that the Turtle output is more textual, where concatenation, newlines and whitespaces need to be considered in the query that perform the mapping. On the other hand, in RDF/XML opening and closing tags as well as nested elements should be taken into account. As a result, XQuery query in RDF/XML syntax will create more triples than the Turtle query. For this reason, the next stage considers representing XQuery query solely in Turtle format.

5.6 XQuery to RDF Mapping

The final stage in the implementation phase and the most important objective of this research is identifying solutions that allow users to translate query-based mappings into an RDF data model. This translation will allow users to run SPARQL queries over the RDF version of the query used in the mapping. SPARQL queries enable the analysis and reasoning processes to be applied over the queries as well as manipulating them.

Many solutions were proposed to convert XQuery into RDF data models as part of this research. Two of the solutions were investigated and discussed in this section, while the other solutions were discarded for various reasons which are outlined at the end of this section. The considered solutions are divided into two different approaches; manual and automatic mapping approaches.

5.6.1 Manual mapping

The first solution towards achieving successful mapping between XQuery query and RDF data model format is based on an initial design undertaken by Alan Meehan, a PhD student within the Knowledge and Data Engineering group. The mapping in this approach is a form of manual mapping, where XQuery queries are translated into an RDF data model based on a pre-defined specification. The proposed name of this design, as given by the creator of the specification is XQR which stands for XQuery as RDF. XQR specification (see Appendix A) contains a number of classes and properties that can be used to model XQuery queries as RDF data model. A few changes have been suggested during this research for the XQR specification in order to enhance the result of the mapping.

In the current version of XQR specification there are approximately ten OWL classes and two properties that can be used to represent an XQuery query in RDF. Classes and properties that model the main parts of XQuery along with their definitions are listed below.

- DeclareClause: this OWL class used to model 'Declare Clause'.
- DeclareVariable: after modeling the declare clause, the variable that is used in declare clause need to be modeled as well. Two Owl properties are used with this OWL class, the first property represents the variable while the second property represent the assigned value to the variable.
- Let clause: this OWL class is used to model let clauses in XQuery. It uses two properties, one for declaring the variable and the second properties for defining the assigned value.
- Return clause: to model the return clause in XQuery this class is used.
- Function: this OWL class is used to represent the functions used in XQuery. This class also uses two properties, which are the function name property (to reflect the name of the function) and the arbitrary number of assigned argument.

The first stage in applying this manual mapping is reading and understanding all of the classes and properties listed in the specification. The classes and properties defined in the specification are then used to represent the XQuery query in RDF. At this stage, the triples in this manual mapping are defined as blank nodes. Each triple has a unique identifier which can be used multiple times within the mapping document. Blank nodes introduce some limitations which will be discussed in the Evaluation chapter.

The result of mapping the XQuery FLOWR expression (see Figure 5.4) which runs over the "List.xml" document and converts it into Turtle format is presented in Appendix B. Full analysis of the manual mapping output and the suggested changes to the XQR specification are presented in the Evaluation chapter.

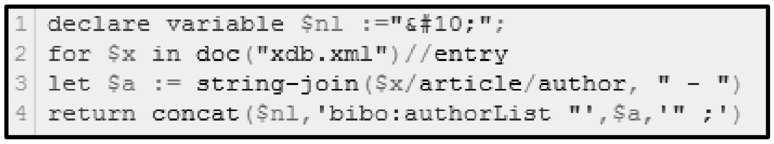
5.6.2 Automatic mapping

The second proposed solution focused on automating the translation process. XQueryX has been suggested as an intermediate step in this project which could contribute to automating the mapping process, produce more efficient results and minimize mapping efforts. However, in past few years, XQueryX has been suggested as a method to define an XML representation for XQuery queries. Thus, representing XQuery in an XML syntax will make it possible to use the wide variety of tools available and converters that manipulate XML documents.

At this stage, XQuery query is translated into XQueryX [20]. Then, XQueryX is translated into RDF by using the available XML to RDF converter. This automatic mapping can be divided into two mapping stages. The first mapping converts an XQuery query to XQueryX, while the second mapping converts XQueryX (which is the output from the previous stage) to an RDF data model.

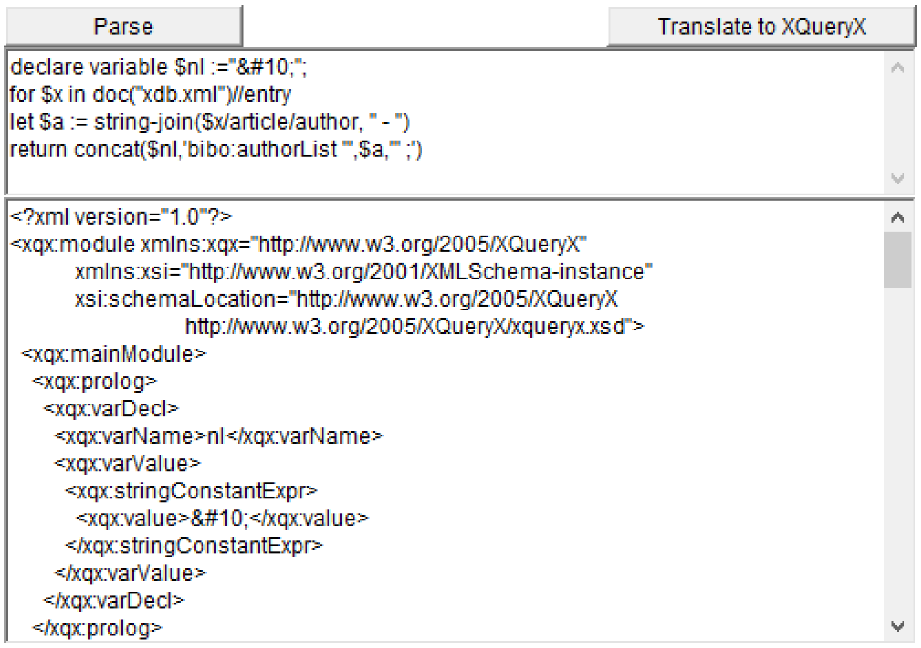
5.6.2.1 XQuery to XQueryX mapping

A limited number of implementations are available for XQueryX. One of the available implementations was adopted in this research to convert an XQuery query into an XML representation. This implementation could be described as an online W3C applet-based tool that enables users to translate XQuery expressions to XQueryX. As mentioned in [39], the parser performs its parsing process based on the XQuery grammar that is available in the page as JavaCC/JJTree files. Moreover, it produces an error if the syntax of the input is incorrect. For the reason that this implementation of XQueryX generates a very long representation, only one snippet (see Figure 5.8) of the mapping from the original query (in Figure 5.4) is translated into XQueryX. The result of translating the mapping snippet in in Figure 5.8 to XQueryX is presented in Appendix C. However, Figure 5.9 shows the online applet interface which allows users type in an XQuery 1.0 expression and then click the button to translate it to XQueryX. The full XQueryX output for the original query can be found in the attached CD.



```
declare variable $nl :="&#10;";
for $x in doc("xdb.xml")//entry
let $a := string-join($x/article/author, " - ")
return concat($nl,'bibo:authorList "',$a,'" ;')
```

Figure 5.8 - One snippet of mapping (from XML to Turtle).



```
declare variable $nl :="&#10;";
for $x in doc("xdb.xml")//entry
let $a := string-join($x/article/author, " - ")
return concat($nl,'bibo:authorList "',$a,'" ;')
```

```
<?xml version="1.0"?>
<xqx:module xmlns:xqx="http://www.w3.org/2005/XQueryX"
        xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
        xsi:schemaLocation="http://www.w3.org/2005/XQueryX
                    http://www.w3.org/2005/XQueryX/xqueryx.xsd">
 <xqx:mainModule>
  <xqx:prolog>
   <xqx:varDecl>
    <xqx:varName>nl</xqx:varName>
    <xqx:varValue>
     <xqx:stringConstantExpr>
      <xqx:value>&#10;</xqx:value>
     </xqx:stringConstantExpr>
    </xqx:varValue>
   </xqx:varDecl>
  </xqx:prolog>
```

Figure 5.9 - The main interface of XQuery to XQueryX applet-based tool.

5.6.2.2 XQueryX to RDF mapping

After translating XQuery to an XML syntax, an XML to RDF converter is used. Representing XQuery queries in RDF provides users with the ability to query, manipulate and analyze these queries by using SPARQL query language. Numerous tools exist to translate from XML to RDF (see [44]) most of these solutions rely on XSLT stylesheets to perform the mapping. However, in this stage of mapping, two different converters that convert from XML to RDF were used in order to perform the translation process. One of these is based on an XSLT stylesheet and the other one is based on XSPARQL language. The mapping from these two converters are compared and evaluated in Chapter 6.

The first converter is the released implementation of XSPARQL which is discussed in detail in section 3.3.1. XSPARQL allows the translation between XML and RDF in both directions. The implementation is available as an online tool (http://xsparql.deri.org/). Via the website, users can upload XML files and select the RDF output format with different RDF serialization types available. Conversely, RDF files can be uploaded and the output in this case will be an XML file. Since this tool builds the triples with a very long URI, only one of the produced triples is presented. The RDF output of the full XQuery in Figure 5.4 is available in the attached CD.

The triple in Figure 5.10 consists of an object, two predicates and two additional objects. An object is located in the first part of the triple which refers to the return clause in XQuery. Then the first predicate *hasResource* models the relationship between the subject and its first object. The first object refers to the arguments of the return clause. The second predicate and object represent the name of the function used in the return clause along with the actual value of the modeled function which is the *Concat* function.

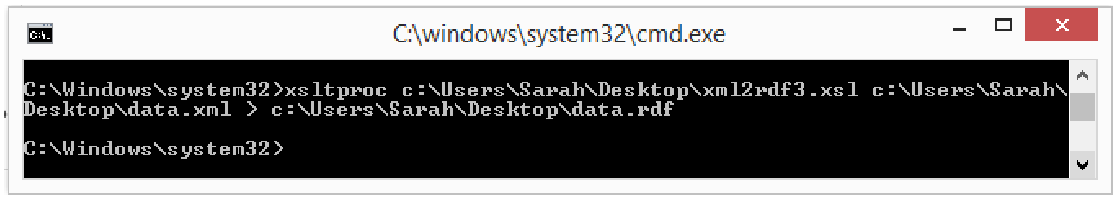
```
<http://x2r.com/Resource/xqx:module/xqx:mainModule/xqx:queryBody/xqx:fl
worExpr/xqx:returnClause/xqx:functionCallExpr>
    hms:hasResource
<http://x2r.com/Resource/xqx:module/xqx:mainModule/xqx:queryBody/xqx:fl
worExpr/xqx:returnClause/xqx:functionCallExpr/xqx:arguments> ;
    <http://x2r.com/Property/xqx:functionName>
         "concat" .
```

Figure 5.10 - Triple in turtle format produced by XSPARQL based converter.

The second converter is a generic XML to RDF converter which uses an XSLT stylesheet to translate an XML file into an RDF data model [50]. This generic converter was design for *Astro-Grid-D* which is a project that enables grid science in the German Astronomical community [51]. In this project, many tasks such as the monitoring of a robotic telescope and computing resources were based on a XML syntax. On the other hand, the information service of main part in the project is based on RDF. Therefore, the translation between the two formats was needed.

In this project, the XQuery query in Figure 5.4 in its XQueryX form was used with the stylesheet to produce the RDF data model. The transformation is executed with an XSLT processor which called XSLTPROC. XSLTPROC is a command line based tool for applying XSLT stylesheets over XML files. The tool is invoked from the command line by one command line as illustrated in Figure 5.11 . The name of the stylesheet should be written first, followed by the name of the XML file which the stylesheet is applied over [52].

For this project, the command line is (xsltproc xml2rdf3.xsl data.xml > data.rdf). xml2rdf3.xsl in the command is the XSLT stylesheet file which can be downloaded from [50]. Data.xml is an XML file which needs to be converted to RDF and contains the XQuery query in Figure 5.4 in its XQueryX form. Finally, data.rdf is an RDF document which holds the output in RDF/XML format. Full analysis of the produced output from the XSPARQL based converter and the XSLT based converter is presented in the Evaluation chapter.



```
C:\Windows\system32>xsltproc c:\Users\Sarah\Desktop\xml2rdf3.xsl c:\Users\Sarah\
Desktop\data.xml > c:\Users\Sarah\Desktop\data.rdf

C:\Windows\system32>
```

Figure 5.11 - Command line to translate XML file into RDF by applying XSLT stylesheet.

5.6.3 Alternative solution

One of the proposed solutions that was not successfully achieved for several reasons was writing an XQuery to perform the mapping from XQueryX to RDF automatically in a generic fashion. This solution was the second proposed solution after the manual mapping. It was intended to be applied by following the same steps of the previous automatic approach.

The solution starts by converting XQuery into XQueryX. Then an XQuery query is written in order to query the XQueryX and construct the RDF output. It was assumed the mapping would be performed based on the predefined vocabulary specification (see Appendix A).

This solution was not successfully achieved for various reasons. First, the available XQueryX implementation creates deeply nested XML documents. This makes it difficult and not feasible to achieve a generic approach by using XQuery, as all the possible XQuery structures should be taken under consideration.

Moreover, based on the current version of the vocabulary specification and the structure of XQuery there is high dependency between the triples. In order to construct an RDF document that reflects the same XQuery structure in an efficient way, all these dependencies and interlinking between the triples should be considered during the mapping process, not afterwards. For example, a triple that reflects the statement of declaring a variable, **X**, would be used in various locations within the RDF document e.g. in the return clause or in defining a path. Furthermore, the subject that refers to variable **X** is a blank node with given identifier. This identifier should be used every time variable **X** is called during the mapping process. Therefore, these identifiers should be tracked or an approach for building IRI should be employed.

For the previously reasons highlighted, the mapping from XQueryX to RDF by using XQuery was deemed to be complicated and therefore not a feasible approach to consider for this type of mapping.

5.7 Summary

In this chapter, the programming languages and other technologies used during the implementation phase have been defined. The implementation phase was presented based on the proposed layered architecture in the Design chapter. Each layer represented one mapping stage, the first layer was the mapping from specific data format into XML. The data format used in this project was Bibtex format. The second layer reflected the mapping from an XML format to RDF based on a query based mapping approach. The third layer consisted of mapping the query that was used in the mapping process in the previous layer to an RDF data model. The proposed solutions to achieve the mapping in the third layer were divided into manual and automatic solutions. In the manual approach, data was uplifted from XML syntax into an RDF data model manually based on a predefined vocabulary specification. The automatic approach focused on translating an XQuery query into an RDF data model based on the available tools. Finally, a proposed solution that was not successfully achieved was discussed along with clarification and logic for its dismissal.

Chapter 6 : Evaluation

6.1 Introduction

The goal of this research has been to ascertain if representing a query-based mapping (XQuery queries) using RDF can allow further analysis and recomposition of the mapping statements. This chapter evaluates the two proposed approaches to convert an XQuery query into an RDF data model, i.e. using the manual and the automatic approaches. The approaches are evaluated according to knowledge required to perform the mapping process, effort involved and completeness of mapping results. Furthermore, the mapping results (RDF representation of XQuery) from the two approaches is analyzed by running a SPARQL query over them. This chapter is broken down as follows: firstly, the manual approach is discussed in detail along with suggestions that could improve it. Secondly, the chapter discusses the automatic approach and its feasibility in converting an XQuery query into RDF. Then, the chapter focusses on the analysis of the mapping results from the two approaches by using a SPARQL query. Finally, a comparison table of the two approaches is presented.

6.2 Manual approach

The manual approach in this project is called XQR. In this mapping process, XQuery queries are translated into an RDF data model, based on a predefined specification, manually. This approach is evaluated based on three criteria, which are knowledge required in order to perform the mapping, involved effort and the completeness of the generated triples. Last but not least, limitations of this approach are discussed in depth.

6.2.1 Required knowledge

In order to perform this manual mapping, there are a set of requirements that should be met. Users shall have a reasonable level of understanding of semantic web technologies as a whole. More precisely, they shall be able to understand and handle RDF in turtle format, OWL classes and properties, XML and XQuery.

- XML

Data in this project is uplifted from XML to RDF using XQuery. However, in order to apply this approach, a reasonable level of understanding is required which must cover the following: XML Schema, how the data is structured within an XML document, order of elements, type of elements, and number of child elements.

- XQuery

The uplifting in this project is performed by applying a query based approach which uses an XQuery query to perform the mapping. XQuery is designed to query XML files as well as anything that can be represented using XML. Users should have a reasonable understanding of the following: how to write simple and complex XQuery queries which allow data uplifting, XQuery FLOWR expressions and XQuery functions which might be used during the mapping.

- OWL

OWL is a language used to express complex knowledge about entities and relationships between them in the semantic web. This manual mapping is based on a vocabulary specification which defines a set of OWL classes and OWL attributes. This specification is used during the mapping in order to model an XQuery query in RDF. Users who will apply this approach shall be able to use OWL classes and attributes when they are building RDF triples.

- RDF (Turtle serialization format)

In this form of mapping, XQuery is modeled in RDF, more precisely in Turtle format. Turtle format is more human-readable than RDF/XML. Therefore, Turtle is more suitable in order to apply the mapping manually. Therefore, an intermediate to advanced level of understanding of RDF triples in turtle syntax is required.

6.2.2 Effort involved

Applying the automatic mapping process required significant cognitive effort, especially when the mapping became more complex. In this research, the emphasis is placed on identifying an appropriate approach that enables the analysis and manipulation of mapping statements. Therefore, the manual mapping was a good starting point before automating the mapping process. This approach helped in identifying some potential issues that could be faced later. Furthermore, it contributed to building a basic infrastructure for the automating mapping approach. When the mapping becomes more complex and more data is involved, it is not a feasible to apply this mapping manually.

6.2.3 Completeness of mapping results

The mapping output (triples) resulting from translating an XQuery query into RDF should be complete and reflect the same structure of the mapped query. In manual mapping, this could be guaranteed because the user who will perform the mapping process manually can verify the completeness of the produced output. Also, there are RDF validation tools available that represent RDF data models in graphical form which allow users to check the completeness of their documents visually.

6.2.4 Limitations

Limitations of this approach include the following:

- Errors

Humans make mistakes. In this approach there is a high possibility of facing syntax and semantic errors. Syntax errors can be checked by using one of the available validators or any SPARQL query engine. On the other hand, semantic errors could emerge because of misunderstanding the specification.

- Incomplete specification

The current XQR specification (see appendix A) used in this research is not complete. More classes and attributes should be added to it in order to express more complex XQuery queries. Currently, only simple queries can mapped into RDF with this specification. Examining how XQueryX represents XQuery could help in extending this specification.

- Blank nodes

In the current specification, blank nodes with identifiers are assigned to the subjects of all triples. Blank nodes in RDF is neither URIs nor literals. Also, blank nodes have only local scope. Using blank nodes in this research be seen as a limitation of this approach especially when triples need to be analyzed or manipulated.

Finding an alternative representation for blank nodes in the sematic web in general is not a trivial task. Blank nodes form part of the RDF specification and core aspect of semantic web technology. Moreover, blank nodes are being used heavily in a vast number of tools and datasets in the web. In [53], authors stated that dealing with the issue of blank nodes is important and is a time-consuming issue. Furthermore, it is a complicated and costly task. Thus, before finding and applying an alternative to replace the use of blank nodes, all of these consequences must be considered. However, authors claimed that any change to the core in the semantics of blank nodes would have a large associated cost and it is not clear what a better alternative would be. The RDF 1.1 Working Group is of the same opinion about not replacing blank nodes, but they discourage overusing or misusing blank nodes. Also, they recommend providing Skolemization schemes which allow mapping blank nodes to global URIs [53][54].

These limitations form the real motivation behind the automatic approach. The automatic approach which is evaluated in the next section needs to address and tackle these shortcomings.

6.3 Automatic approach

In this approach, mapping between XQuery and RDF is applied automatically using different converters available. This approach is evaluated based on three criteria, which are: knowledge required in order to perform the mapping, effort involved and the completeness of the generated triples. Finally, limitations of this approach are too discussed in detail.

6.3.1 Required knowledge

A set of requirements should be met in order to perform this automatic mapping. As with the manual mapping, users who will apply this approach shall have reasonable level of understanding of semantic web technologies. Furthermore, an intermediate to advanced level of understanding of RDF in both Turtle and RDF/XML syntax, XML files and XQuery is required.

- XML

Data in this project is uplifted from XML to RDF using XQuery. However, in order to apply this approach, a reasonable level of understanding is required which must cover the following: XML Schema, how the data is structured within XML document, order of elements, type of elements and number of child elements.

- XQuery

As before with manual mapping, users should have reasonable understanding of the following: how to write simple and complex XQuery queries which allow data uplifting, XQuery FLOWR expressions and XQuery functions which might be used during the mapping.

- RDF (Turtle and RDF/XML serialization formats)

In this mapping, XQuery is modeled in RDF, in both Turtle and RDF/XML formats. Two different converters were used, with one of them producing the desired output in Turtle while the other converter produced it in RDF/XML format.

6.3.2 Effort involved

Applying the automatic mapping required less cognitive effort than manual because of the tools which facilitate automatic mapping. XQuery queries are mapped into RDF using tools over two different stages. In other words, two tools were used sequentially, in which the output from the first tool acted as an input for the second one. The first tool converted XQuery to XQueryX by using an online applet based tool. The output from this applet could then be copied in order to create an XML file to be uploaded to an online tool to convert it to RDF. This was alternatively achieved by applying an XSLT stylesheet to this file using the command line.

6.3.3 Completeness of mapping results

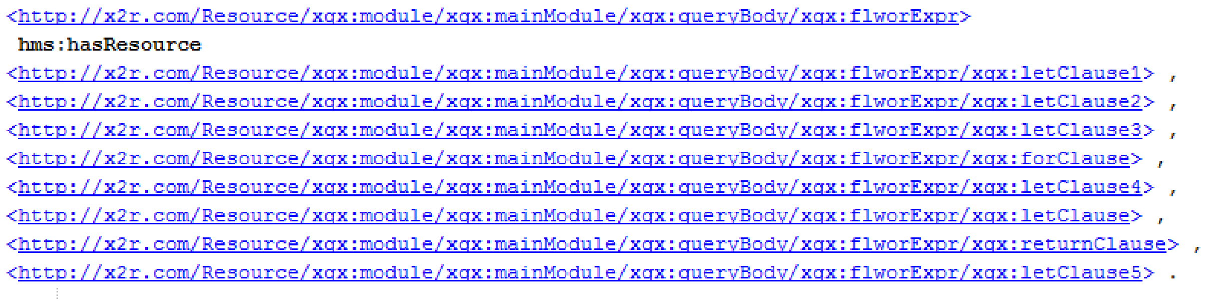
The mapping results in this approach depend entirely on how the XQueryX implementation represents the XQuery. There are two outputs from this mapping (one output from each converter). These results are evaluated separately based on the conversion type.

- XSPARQL based converter

The output of this converter can be expressed in Turtle syntax which makes it easy to read and analyze. The subjects and most of the predicates, as well as objects in this converter are expressed using URIs. This XSPARQL based converter build URIs based on the structure of an XML file. Therefore, it provides fully descriptive triples. For this reason, it is easy to recognize which part of an XQuery is being expressed from the name of the subject. Conversely, in the manual approach, the subject is represented using blank nodes which indicate nothing about the triple. Using these URIs that reflect the structure of an XQuery query file could be considered as an alternative for blank nodes in the manual approach.

Furthermore, this XSPARQL converter created a complete set of triples that expressed all XQuery query parts and linked them together. Figure 6.1 shows an example of how all parts of XQuery are linked. In this triple, the subject reflects the type of XQuery query

(FLWOR expression). This subject has eight objects which are: let clause 1, let clause 2, let clause 3, let clause 4, let clause 5, for clause and return clause, which are the main parts in the mapped XQuery query.



```
<http://x2r.com/Resource/xqx:module/xqx:mainModule/xqx:queryBody/xqx:flworExpr>
 hms:hasResource
<http://x2r.com/Resource/xqx:module/xqx:mainModule/xqx:queryBody/xqx:flworExpr/xqx:letClause1> ,
<http://x2r.com/Resource/xqx:module/xqx:mainModule/xqx:queryBody/xqx:flworExpr/xqx:letClause2> ,
<http://x2r.com/Resource/xqx:module/xqx:mainModule/xqx:queryBody/xqx:flworExpr/xqx:letClause3> ,
<http://x2r.com/Resource/xqx:module/xqx:mainModule/xqx:queryBody/xqx:flworExpr/xqx:forClause> ,
<http://x2r.com/Resource/xqx:module/xqx:mainModule/xqx:queryBody/xqx:flworExpr/xqx:letClause4> ,
<http://x2r.com/Resource/xqx:module/xqx:mainModule/xqx:queryBody/xqx:flworExpr/xqx:letClause> ,
<http://x2r.com/Resource/xqx:module/xqx:mainModule/xqx:queryBody/xqx:flworExpr/xqx:returnClause> ,
<http://x2r.com/Resource/xqx:module/xqx:mainModule/xqx:queryBody/xqx:flworExpr/xqx:letClause5> .
```

Figure 6.1 - RDF Triple represents all XQuery query parts

- XSLT based converter

This is an alternative converter that was used to convert an XQuery expressed in XML to RDF. This generic converter produced an output in RDF/XML formats as a result of applying an XSLT stylesheet to an XML file. Furthermore, for this simple form of mapping, this converter created a very deeply nested XML file which contained over 1000 lines. Therefore, the output from this converter is not useful for the purpose of this research and is hence infeasible, since it depends upon an XSLT stylesheet.

6.3.4 Limitations

Limitations of this approach include the following:

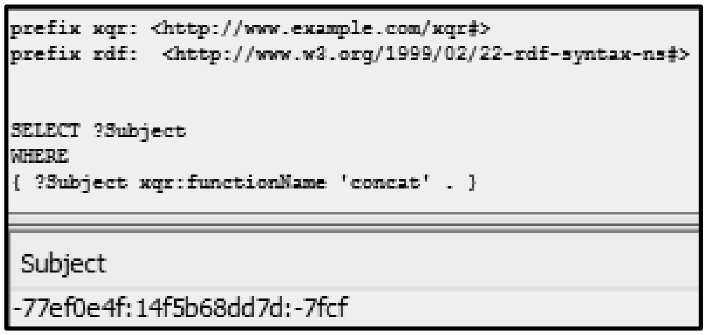
- The main issue with this approach is the dependency upon other available tools. For instance, the XSPARQL implementation used in this research is an online tool that allows users to upload their XML files and obtain the corresponding RDF files. The availability of this online tool is not guaranteed.
- The converter based on XSLT created deeply nested triples and most of them were meaningless, hence it is not useful for the purpose of this research. On the other hand, the converter based on XSPARQL creates a complete set. Also the number of

the triples resulting from this conversion method are fewer than the number of triples produced by the XSLT stylesheet based tool.

6.4 Analysis of the mapping results (SPARQL)

The main goal of this research was to identify approaches that allow the expression of query based mapping in RDF format in order to analyze and manipulate these mapping statements. SPARQL query language is used to analyze the RDF form of an XQuery query. In this stage of the evaluation, mapping results from the manual and automatic approaches were both queried. These results from both approaches are evaluated and compared. This section presents two examples of the applied SPARQL queries along with their results. The queries are first applied over the output from the manual approach and then over the output from the automatic approach to show the differences between them.

The query was first applied in order to analyze the RDF document, designed to selects the subject of a specific function name. In Figure 6.2, the query is written to select the subject name of the function *concat* from the manual approach mapping result. The result of this SPARQL query as shown in the figure is **(-77ef0e4f:14f5b68dd7d:-7fcf)** which is a randomly generated string. This query illustrates the main issue with using blank nodes to represent the subject in triples. In this case, a second query is needed to identify the type of subject or to investigate it.

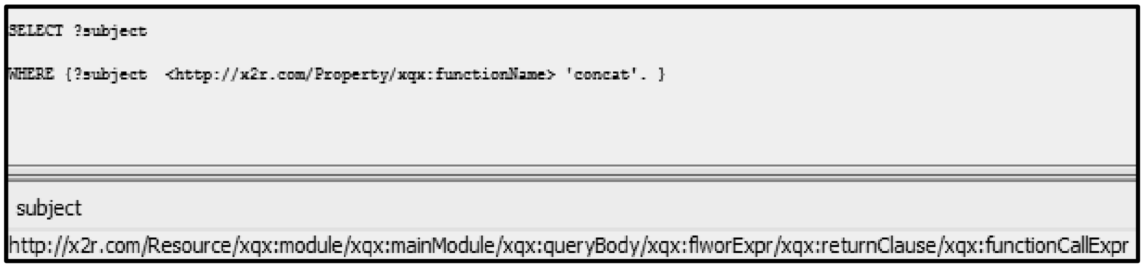


```
prefix xqr: <http://www.example.com/xqr#>
prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>


SELECT ?Subject
WHERE
{ ?Subject xqr:functionName 'concat' . }
```

```
Subject
-77ef0e4f:14f5b68dd7d:-7fcf
```

Figure 6.2 - SPARQL query to select a subject of a triple (manual approach).

In Figure 6.3, the same query is applied over the mapping result from the automatic approach (based on XSPARQL). The result of this SPARQL query as shown in the figure is (http://x2r.com/Resource/xqx:module/xqx:mainModule/xqx:queryBody/xqx:flworExpr/xqx:returnClause/xqx:functionCallExpr) which provides more details about the concat function, such as the location of this function within the XQuery query. As illustrated in the subject URI, the concat function is part of the return clause and the return clause is part of the FLWOR expression.

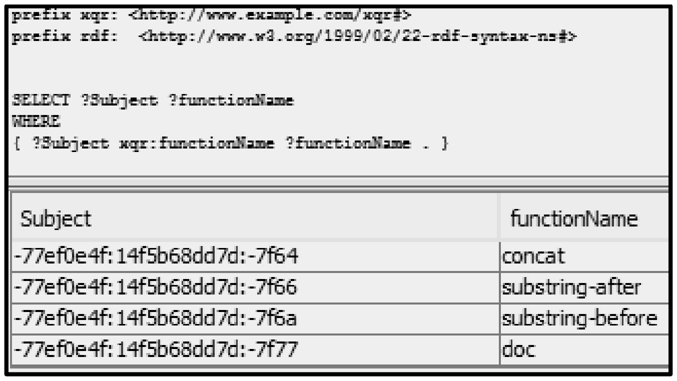


```
SELECT ?subject

WHERE {?subject  <http://x2r.com/Property/xqx:functionName> 'concat'. }
```

subject
http://x2r.com/Resource/xqx:module/xqx:mainModule/xqx:queryBody/xqx:flworExpr/xqx:returnClause/xqx:functionCallExpr

Figure 6.3 - SPARQL query to select a subject of a triple (automatic approach).

A second query was applied to select all the function names in the document along with their subjects. As shown in Figure 6.4, the query is written to select the subject of all functions as well as the function name from the manual approach mapping result. The result of this SPARQL query as shown in the figure shows all function names which are: *concat, substring-after, substring-before* and *doc*. Also, this query lists the subject of these functions which are blank nodes with randomly generated identifiers.

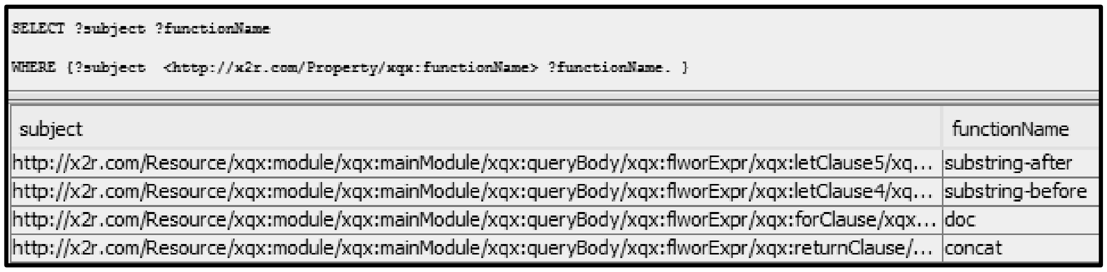


```
prefix xqr: <http://www.example.com/xqr#>
prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>


SELECT ?Subject ?functionName
WHERE
{ ?Subject xqr:functionName ?functionName . }
```

Subject	functionName
-77ef0e4f:14f5b68dd7d:-7f64	concat
-77ef0e4f:14f5b68dd7d:-7f66	substring-after
-77ef0e4f:14f5b68dd7d:-7f6a	substring-before
-77ef0e4f:14f5b68dd7d:-7f77	doc

Figure 6.4 - SPARQL query of all function names (manual approach).

In Figure 6.5, the same query is shown as applied over the mapping result from the automatic approach (based on XSPARQL). The result of this SPARQL query, as displayed in the figure, lists all the function names in the RDF file. It can be observed that subjects here are more descriptive, as it describes the location of each function according to the XQuery query structure. For instance, the substring-after function is used in the fifth let clause.



```
SELECT ?subject ?functionName

WHERE {?subject  <http://x2r.com/Property/xqx:functionName> ?functionName. }
```

subject	functionName
http://x2r.com/Resource/xqx:module/xqx:mainModule/xqx:queryBody/xqx:flworExpr/xqx:letClause5/xq...	substring-after
http://x2r.com/Resource/xqx:module/xqx:mainModule/xqx:queryBody/xqx:flworExpr/xqx:letClause4/xq...	substring-before
http://x2r.com/Resource/xqx:module/xqx:mainModule/xqx:queryBody/xqx:flworExpr/xqx:forClause/xqx...	doc
http://x2r.com/Resource/xqx:module/xqx:mainModule/xqx:queryBody/xqx:flworExpr/xqx:returnClause/...	concat

Figure 6.5 - SPARQL query of all function names (automatic approach).

6.5 Comparison between the two approaches

The following table summarizes the main differences and compares the two approaches based on number of triples in the file that contains the result of the XQuery query to RDF data model conversion, the size of this file, required knowledge in order to apply the approaches and effort involved.

Table 6-1 Comparison between manual and automatic approaches.

Approach	**Total number of triples**	**File size**	**Required knowledge**	**Effort involved**
Manual	104	2.79 KB	XML XQuery RDF OWL	More cognitive effort.
Automatic	193	39.8 KB	XML XQuery RDF	Less cognitive effort.

In order to show the differences between the two approaches, the same XQuery was translated into RDF by using the two approaches independently. It can be seen from the values in Table 6-1 that the total number of triples produced in the automatic approach is nearly double the number of triples produced in the manual approach. The main reason for this is that the automatic approach creates triples based on how XQueryX expressed an XQuery query in XML representation. The current implementation of XQuery created very deeply nested XML documents which eventually resulted in the creation of more triples with a long URI (URI in the automatic approach is being built according to the XML file hierarchy). The same reason also causes the significant difference in the produced file size.

6.6 Encountered difficulties

Two main difficulties were encountered during the evaluation phase which result in limiting the evaluation process. One of these difficulties is the unavailability of XSPARQL online based implementation. This tool became unavailable after receiving one result from it. “Coming soon” was written in their online website (http://xsparql.deri.org/). Because of this issue only one result of one query form XSPARQL implementation was evaluated against the manual approach result. Thus, evaluating the mapping of more complex query was not possible.

The other difficulty encountered during the evaluation is finding and configuring an appropriate tool that support SPARQL 1.1. This version of SPARQL supports more sophisticated operations such as insertion and deletion. Because of this only select queries were written in order to evaluate the RDF files by using Twinkle which is a tool that supports only the older version of SPARQL. Thus, only analyzing the RDF data model of an XQuery query was achieved. However, manipulating these queries by using SPARQL 1.1 could be investigated as part of future work.

6.7 Summary

In this chapter, both approaches have been evaluated according to various factors. The limitations of each approach are discussed in detail. The XQuery in an RDF data model was queried using a SPARQL query language. A comparison table of both approaches has been provided. On the basis of the results and data collected, it can be concluded that both the manual and automatic (XSPARQL based) approaches can successfully create complete triples which modeled a simple XQuery query in RDF data model format. The specification of the manual approach needs to be extended in order to enable the mapping of more complex XQuery queries. Furthermore, it was observed that the automatic approach has an advantages over the manual approach in creating complete, meaningful triples and minimizing the required effort in order to perform the mapping.

Chapter 7 : Conclusion

7.1 Introduction

In this chapter, the main contribution of this project and conclusions are discussed. This leads onto a proposal of optional future work which would further benefit the research, and concludes with final remarks.

7.2 Conclusions

The main objective of this research was to identify solutions that allow users to translate query-based mappings into RDF based data models. Two approaches were explored in this report in order to provide a solution for this objective, i.e. manual and automatic mapping approaches. Both approaches were deemed suitable for converting an XQuery query into RDF data model.

Investigating the accuracy and the issues that arise in translating an XQuery into RDF, both automatically and manually, was also one of the research objectives. The accuracy in these mapping approaches pertains to the results of the mapping. The completeness of the triples that reflect the same structure of an XQuery query can be considered to be an accuracy measurement of the mapping result. The completeness of triples in the manual approach depends entirely on the user who will perform the mapping. After the manual mapping is performed, an RDF validator can be used to validate and visualize the output of this mapping. However, the automatic approach based on XSPARQL implementation produces complete sets of triples which reflect the mapped XQuery query structure.

Most of the issues which arose in the manual mapping were related to the current specification. Because of human error, applying the mapping process manually is not feasible, especially with more complex mapping statements. Furthermore, there is a high possibility that syntax and semantic errors may appear in this type of mapping. However, the main issue with the automatic approach is the dependency on other online mapping tools in order to perform the mapping and the fact that the availability of these tools cannot guaranteed. Furthermore, the output of this mapping contains more triples than the manual mapping output due to the use of XQueryX as an intermediate step. This resulted in a significant difference between file sizes.

Another objective of this research was to evaluate the two approaches by comparing them. The automatic and manual mapping processes can be used for translating an XQuery query to an RDF data model. Currently, manual mapping can be used to translate only simple XQueries. The XQR specification in the manual mapping need to be extended in order to allow for more complex mapping. The way that XQueryX represents XQuery in XML is helpful for identifying what needs to be added to the XQR specification. Moreover, the automatic approach creates meaningful URIs based on the XML representation of an XQuery query in order to represent the subject in a triple. Thus, from such a URI, further information about the triple can be extracted. Conversely, the manual approach uses blank nodes to represent a subject in a triple, which reveals nothing about the triple.

Finally, the research question was addressed by querying the RDF data model of the XQuery query. It has been proven that representing an XQuery query in RDF will allow further analysis of the query. In this research, the automatic approach gives better results than the mapping approach when their outputs are queried using SPARQL query language.

To conclude, this research successfully proposed two different approaches which contribute in identifying solutions to represent an XQuery query in an RDF data model. However, users have been increasingly uplifting data into RDF data models. Thus, there is a need for such a system that can create, analyze and manipulate these mappings automatically.

7.3 Future work

Potential related aspects of this research have been considered, which could be investigated as part of future work.

The limitation of the two approaches can be tackled in future work following on from this study. In the case of manual mapping, the XQR needs to be extended in order to allow the representation of more complex mapping. Furthermore, blank nodes should be avoided during the construction of triples. Blank nodes are easier for the producers to create, but it puts a burden on data consumers. Thus, it is not suitable for the purpose of this research because data will be analyzed and modified. After tackling these issues, this specification can be used to build a parser which can perform the mapping process automatically. Alternatively, the automatic approach would be more efficient if the number of the produced triples could be reduced. Moreover, this study uses only Bibtex formatted data as a use case domain. Future work could investigate the feasibility of these approaches with more complex use case domains, which is one of the research objectives that was not considered due to the time constraints for this study.

7.4 Final remarks

People are increasingly uplifting data into RDF format, which is a standard model for data interchange on the web. However, because data schemas and ontologies that are used to describe data in RDF may change frequently over time, the need for an approach to represent the mapping statements in an analyzable form has arisen. This dissertation has proposed two approaches to perform the mapping from XQuery to RDF, which are manual and automatic mapping approaches. The two approaches are based on one solution path that begins with the extraction of XML documents using XQuery to uplift data into the standard RDF data model. Then, the XQuery that is used for the mapping process itself is then be transformed into RDF. The overall inspiration for this research is that representing an XQueryX in RDF data model will allow further reasoning and analysis of the query.

Bibliography

[1] A. Dimou, M. Vander Sande, J. Slepicka, P. Szekely, E. Mannens, C. Knoblock, and R. Van De Walle, “Mapping Hierarchical Sources into RDF Using the RML Mapping Language,” *2014 IEEE Int. Conf. Semant. Comput.*, pp. 151–158, 2014.

[2] D. D. Aglio, A. Polleres, N. Lopes, and S. Bischof, “Querying the Web of Data with XSPARQL 1 . 1,” pp. 113–118, 2014.

[3] “RDF - Semantic Web Standards.” [Online]. Available: http://www.w3.org/RDF/. [Accessed: 03-Aug-2015].

[4] L. Yu, *A Developer’s Guide to the Semantic Web*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2011.

[5] “The World Wide Web: A very short personal history.” [Online]. Available: https://www.w3.org/People/Berners-Lee/ShortHistory.html. [Accessed: 02-Aug-2015].

[6] T. Heath and C. Bizer, *Linked Data: Evolving the Web into a Global Data Space*, 1st editio. Morgan & Claypool, 2011.

[7] C. Bizer, T. Heath, and T. Berners-Lee, “Linked data-the story so far,” *Int. J. Semant. Web Inf. Syst.*, vol. 5, no. 3, pp. 1–22, 2009.

[8] “Semantic Web - W3C.” [Online]. Available: http://www.w3.org/standards/semanticweb/. [Accessed: 02-Aug-2015].

[9] “Tutorial 2: Introducing RDF.” [Online]. Available: http://www.linkeddatatools.com/introducing-rdf-part-2. [Accessed: 03-Aug-2015].

[10] “RDF 1.1 XML Syntax.” [Online]. Available: http://www.w3.org/TR/rdf-syntax-grammar/. [Accessed: 03-Aug-2015].

[11] J. de Bruijn, M. Kerrigan, U. Keller, H. Lausen, and J. Scicluna, *Modeling Semantic Web Services: The Web Service Modeling Language*. Springer Science & Business Media, 2008.

[12] “RDF 1.1 Turtle.” [Online]. Available: http://www.w3.org/TR/turtle/. [Accessed: 03-Aug-2015].

[13] E. Prud’hommeaux and A. Seaborne, “SPARQL Query Language for RDF,” *W3C Rec*, 2008. [Online]. Available: http://www.w3.org/TR/rdf-sparql-query/. [Accessed: 28-May-2015].

[14] K. Nguyen, R. Ichise, and B. Le, “Interlinking Linked Data Sources Using a Domain-Independent System,” *Semant. Technol.*, pp. 113–128, 2013.

[15] W. Akhtar, J. Kopecký, T. Krennwallner, and A. Polleres, “XSPARQL: Traveling between the XML and RDF worlds - And avoiding the XSLT pilgrimage,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5021 LNCS, no. X, pp. 432–447, 2008.

[16] “Semantic Annotations for WSDL and XML Schema.” [Online]. Available: http://www.w3.org/TR/sawsdl/#Terminology. [Accessed: 06-Aug-2015].

[17] N. Lopes, S. Bischof, O. Erling, A. Polleres, A. Passant, D. Berrueta, A. Campos, J. Euzenat, K. Idehen, S. Decker, S. Corlosquet, J. Kopecký, J. Saarela, T. Krennwallner, D. Palmisano, and M. Zaremba, “RDF and XML: Towards a Unified Query Layer,” *W3C Work. — RDF Next Steps*, pp. 1–5, 2010.

[18] M. Ferdinand, C. Zirpins, and D. Trastour, “Lifting XML Schema to OWL,” *Web Eng.*, no. 3140, pp. 354–358, 2004.

[19] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, and R. Van De Walle, “RML : A Generic Language for Integrated RDF Mappings of Heterogeneous Data,” 2001.

[20] J. Melton and S. Muralidhar, “XML Syntax for XQuery 1.0 (XQueryX) (Second Edition),” *W3C Rec*, 2010. [Online]. Available: http://www.w3.org/TR/xqueryx/. [Accessed: 28-May-2015].

[21] “R2RML: RDB to RDF Mapping Language.” [Online]. Available: http://www.w3.org/TR/r2rml/. [Accessed: 01-Mar-2015].

[22] F. Michel, J. Montagnat, and C. Faron-zucker, “A survey of RDB to RDF translation approaches and tools.”

[23] L. Han, T. Finin, C. Parr, J. Sachs, and A. Joshi, “RDF123: From spreadsheets to RDF,” *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5318 LNCS, pp. 451–466, 2008.

[24] M. Fiorelli, T. Lorenzetti, and M. T. Pazienza, “Sheet2RDF: A Flexible and Dynamic Spreadsheet Import & Lifting Framework for RDF,” vol. 9101, pp. 131–140, 2015.

[25] “RML Generic Mapping Language.” [Online]. Available: http://semweb.mmlab.be/rml/. [Accessed: 06-Aug-2015].

[26] “RDF Mapping Language (RML).” [Online]. Available: http://semweb.mmlab.be/rml/spec.html. [Accessed: 06-Aug-2015].

[27] "R2RML: RDB to RDF Mapping Language." [Online]. Available: http://www.w3.org/2001/sw/rdb2rdf/r2rml/#triples-map. [Accessed: 05-May-2015].

[28] F. Scharffe, G. Atemezing, and R. Troncy, "Enabling Linked Data Publication with the Datalift Platform," *Work. ...*, pp. 25–30, 2012.

[29] T. R. Gruber and T. R. Gruber, "A translation approach to portable ontology specifications," *Knowl. Acquis.*, vol. 5, no. 2, pp. 199–220, 1993.

[30] "Ontologies - W3C." [Online]. Available: http://www.w3.org/standards/semanticweb/ontology. [Accessed: 22-Aug-2015].

[31] F. Scharffe and D. Fensel, "Correspondence patterns for ontology alignment," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 5268 LNAI, pp. 83–92, 2008.

[32] J. Euzenat and P. Shvaiko, *Ontology Matching*. Berlin, Heidelberg: Springer Berlin Heidelberg, 2013.

[33] "EDOAL: Expressive and Declarative Ontology Alignment Language." [Online]. Available: http://alignapi.gforge.inria.fr/edoal.html. [Accessed: 23-May-2015].

[34] "XSL Transformations (XSLT) Version 2.0." [Online]. Available: http://www.w3.org/TR/xslt20/. [Accessed: 06-Aug-2015].

[35] S. Bischof, S. Decker, T. Krennwallner, N. Lopes, and A. Polleres, "Mapping between RDF and XML with XSPARQL," *J. Data Semant.*, vol. 1, pp. 147–185, 2012.

[36] S. Boag, D. Chamberlin, M. F. Fernández, D. Florescu, J. Robie, and J. Siméon, "XQuery 1.0: An XML Query Language (Second Edition)," *W3C Rec*, 2010. [Online]. Available: http://www.w3.org/TR/xquery/. [Accessed: 28-May-2015].

[37] S. Bischof, N. Lopes, and A. Polleres, "Improve efficiency of mapping data between XML and RDF with XSPARQL," *Lect. Notes Comput. Sci. (including Subser. Lect. Notes Artif. Intell. Lect. Notes Bioinformatics)*, vol. 6902 LNCS, pp. 232–237, 2011.

[38] P. Walmsley, "XQuery," *O'Reilly Media, Inc.*, Apr. 2007.

[39] "XQuery 1.0 Grammar Test Page." [Online]. Available: http://www.w3.org/2007/01/applets/xqueryApplet.html. [Accessed: 05-Aug-2015].

[40] "Numerical Algorithms Group." [Online]. Available: http://www.nag.co.uk/. [Accessed: 05-Aug-2015].

[41] “XQ2XML: XML syntaxes for XQuery.” [Online]. Available: http://monet.nag.co.uk/xq2xml/#s1. [Accessed: 05-Aug-2015].

[42] “Extensible Markup Language (XML).” [Online]. Available: http://www.w3.org/XML/. [Accessed: 14-Aug-2015].

[43] “BibTeX.” [Online]. Available: http://www.bibtex.org/. [Accessed: 14-Aug-2015].

[44] “ConverterToRdf - W3C Wiki.” [Online]. Available: http://www.w3.org/wiki/ConverterToRdf. [Accessed: 06-Aug-2015].

[45] A.-C. Achilles and P. Ortyl, “The Collection of Computer Science Bibliographies.” [Online]. Available: http://liinwww.ira.uka.de/bibliography/index.html. [Accessed: 14-Aug-2015].

[46] F. Jürgen, “Managing Citations and Your Bibliography with BibTEX,” *PracTeX*, no. 4, pp. 1–19, 2007.

[47] “BibTeX as XML markup.” [Online]. Available: http://bibtexml.sourceforge.net/. [Accessed: 14-Aug-2015].

[48] “BibTeXML download | SourceForge.net.” [Online]. Available: http://sourceforge.net/projects/bibtexml/. [Accessed: 14-Aug-2015].

[49] “‘The Bibliographic Ontology.’” [Online]. Available: http://bibotools.googlecode.com/svn/bibo-ontology/trunk/doc/index.html. [Accessed: 15-Jul-2015].

[50] “AstroGrid-D: XML2RDF, an XML to RDF converter.” [Online]. Available: http://www.gac-grid.org/project-products/Software/XML2RDF.htmlhttp:/sourceforge.net/projects/xmltordf/index.html;jsessionid=81B177052978FDD74A3D1C5F2C08EAA2. [Accessed: 17-Aug-2015].

[51] “AstroGrid-D.” [Online]. Available: http://www.gac-grid.org/index.html. [Accessed: 17-Aug-2015].

[52] “xsltproc.” [Online]. Available: http://xmlsoft.org/XSLT/xsltproc.html. [Accessed: 17-Aug-2015].

[53] A. P. Aidan Hogan , Marcelo Arenas , Alejandro Mallea, “Everything you always wanted to know about Blank Nodes,” *Prepr. Submitt. to Elsevier*, vol. 43, no. 2, 2015.

[54] M. Arenas, M. Consens, and A. Mallea, “Revisiting Blank Nodes in RDF to Avoid the Semantic Mismatch with SPARQL,” 2008.

Appendix A: XQR specification

```
@prefix dc: <http://purl.org/dc/elements/1.1/>.
@prefix gic: <http://www.cngl.ie/ontologies/gicup#>.
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>.
@prefix xsd: <http://www.w3.org/2001/XMLSchema#>.
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>.
@prefix owl: <http://www.w3.org/2002/07/owl#>.
@prefix prov: <http://www.w3.org/ns/prov#>.
@prefix cnt: <http://www.w3.org/2011/content#>.
@prefix str: <http://nlp2rdf.lod2.eu/schema/str>.
@prefix sioc: <http://rdfs.org/sioc/ns#>.
@prefix nif: <http://persistance.uni-leipzig.org/nlp2rdf/ontologies/nif-core#>.
@prefix sso: <http://nlp2rdf.lod2.eu/schema/sso#>.
@prefix xqr: <>.

# XQR - XQuery as RDF - Vocabulary Specification .

dc:description  "This vocabulary contains a number of classes and properties that can be used to model XQueries as RDF data."

owl:versionInfo  "0.1" .
dc:date  "2014-07-21" .
dc:creator  "Alan Meehan" .
```

```
## Classes
xqr:XqueryData  a  owl:Class ;
rdfs:label  "XQueryData" ;
rdfs:comment   "This is the parent class of all other classes in the XQR vocabulary. All
other classes are subclasses of this class. " .

xqr:Xquery  a  owl:Class ;
rdfs:subClassOf  xqr:XqueryData ;
rdfs:label  "XQuery" ;
rdfs:comment   "An XQuery. All other parts of the query are ultimately related to this
class. " .

xqr:DeclareClause  a  owl:Class ;
rdfs:subClassOf  xqr:XqueryData ;
rdfs:label  "Declare Clause" ;
rdfs:comment  "Used to model an XQuery 'declare clause'. " .

xqr:DeclareVariable  a  owl:Class ;
rdfs:subClassOf  xqr:XqueryData ;
rdfs:label  "Declare Clause Variable" ;
rdfs:comment   "Used to model a variable that is declared in an XQuery 'declare clause'.
This class uses two properties - xqr:arg1 and xqr:arg2. xqr:arg1 is a type xqr:Variable and
xqr:arg2 is what is being assigned to the variable of xqr:arg1. " .

xqr:ReturnClause  a  owl:Class ;
rdfs:subClassOf  xqr:XqueryData ;
rdfs:label  "return Clause" ;
rdfs:comment  "Used to model an XQuery 'return clause'. " .
```

xqr:LetClause a owl:Class ;

rdfs:subClassOf xqr:XqueryData ;

rdfs:label "Let Clause" ;

rdfs:comment "Used to model an XQuery 'let clause'. This class uses two properties - xqr:arg1 and xqr:arg2. xqr:arg1 is a type xqr:Variable and xqr:arg2 is what is being assigned to the variable of xqr:arg1. " .

xqr:Variable a owl:Class ;

rdfs:subClassOf xqr:XqueryData ;

rdfs:label "Variable" ;

rdfs:comment "Used to model an XQuery 'variable'. This class uses one property - xqr:arg1. xqr:arg1 is used to indicate the name of the variable. " .

xqr:Function a owl:Class ;

rdfs:subClassOf xqr:XqueryData ;

rdfs:label "Function" ;

rdfs:comment "Used to model an XQuery 'function'. This class uses the xqr:functionName property and an arbitrary number of 'xqr:arg' properties. " .

xqr:Path a owl:Class ;

rdfs:subClassOf xqr:XqueryData ;

rdfs:label "Path" ;

rdfs:comment "Used to model XQuery 'path'. It can have an arbitrary number of 'xqr:arg' properties. " .

```
xqr:PathAttribute  a  owl:Class ;
rdfs:subClassOf  xqr:XqueryData ;
rdfs:label  "Path Attribute" ;
rdfs:comment   "Used to model XQuery 'attribute' when it appears in a 'path' e.g.
path1/path2/path3/@attribute1. This class uses one property - xqr:arg1. xqr:arg1 is used
to indicate the name of the attribute.  " .

## Properties
xqr:arg  a  owl:ObjectProperty;
rdfs:domain  xqr:XqueryData ;
rdfs:label "arg";
rdfs:comment "This property can have number appended to it (xqr:arg1, xqr:arg6) and is
used to model all data associated with an instance of xqr:XqueryData, and also model the
order that data appears. ".

xqr:functionName  a  owl:ObjectProperty;
rdfs:domain xqr:Function ;
rdfs:label "Function Name";
rdfs:comment "This property is used to indicate a function's name, such as 'concat',
'data', 'doc' etc. ".
```

Appendix B: The result of mapping XQuery to Turtle

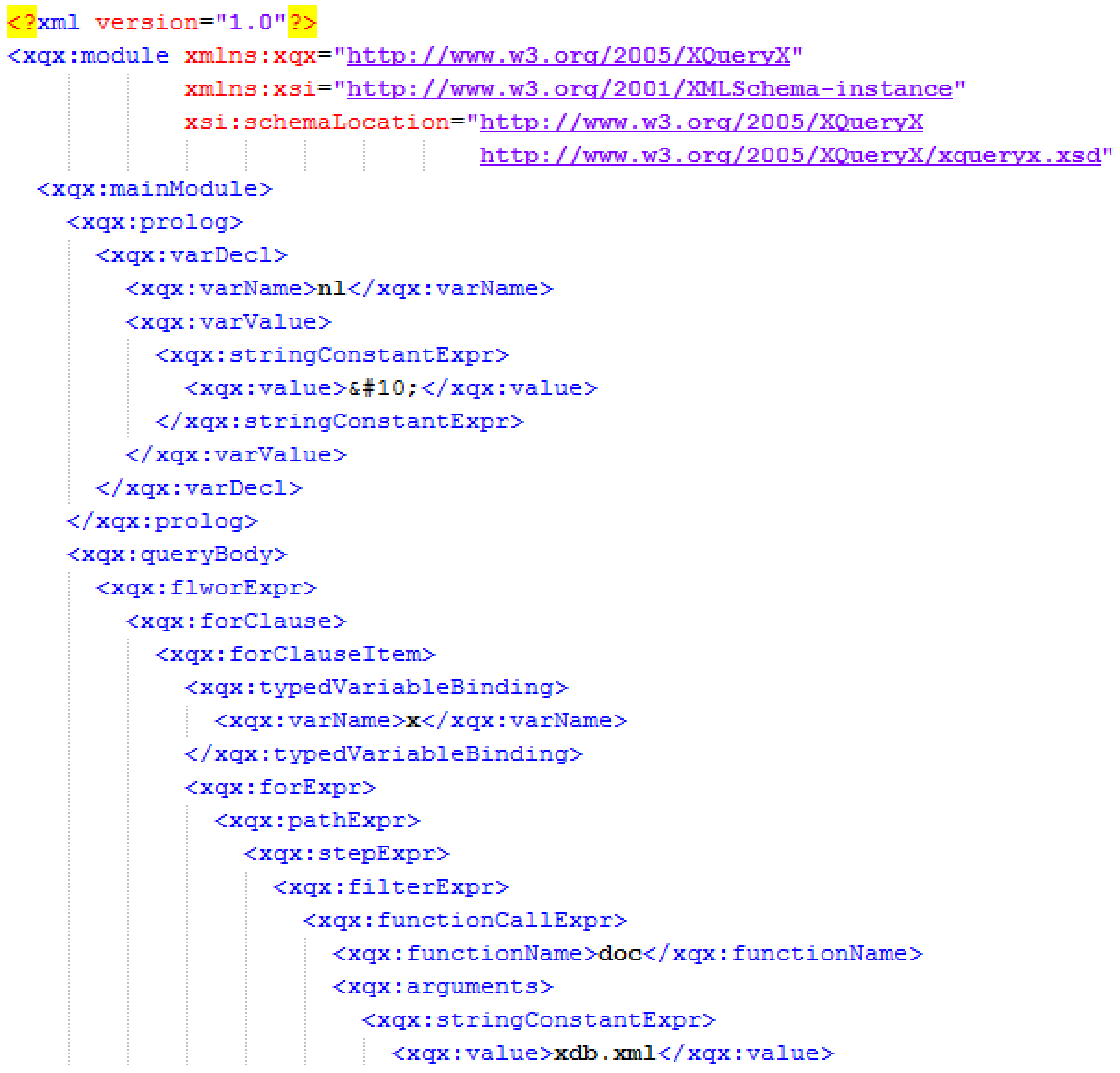
```
@prefix xqr: <http://www.example.com/xqr#>.
@prefix bibo: <http://purl.org/ontology/bibo/> .
@prefix dcterms: <http://purl.org/dc/terms/> .

# XQR REPRESENTATION


### DECLARE CLAUSE 1
_:b1  a  xqr:DeclareClause ;
         xqr:arg1  _:b2 .
_:b2  a  xqr:DeclareVariable ;
         xqr:arg1  _:b3 ;
         xqr:arg2  _:b4 .
_:b3  a  xqr:Variable ;
         xqr:arg1  "nl" .
_:b4  a  xqr:Variable ;
         xqr:arg1  "&#10;" .

### FOR CLAUSE
_:b5  a  xqr:ForClause ;

         xqr:arg1  _:b6 ;

         xqr:arg2  _:b7 .

_:b6  a  xqr:Variable ;

         xqr:arg1  "x" .

_:b7 a  xqr:Function ;

         xqr:functionName  "doc" ;

         xqr:arg1  "xdb.xml" .
```

```
### LET CLAUSE 1
_:b8 a  xqr:LetClause ;
          xqr:arg1  _:b9 ;
          xqr:arg2  _:b10 .
_:b9  a  xqr:Variable ;
          xqr:arg1  "a" .
_:b10 a  xqr:Function ;
          xqr:functionName  "string-join" ;
          xqr:arg1  _:b11 ;
          xqr:arg2  '-' .
_:b11  a  xqr:Path ;
          xqr:arg1  _:b6 ;
          xqr:arg2  "article" ;
          xqr:arg3  "author" .

### LET CLAUSE 2
_:b12  a  xqr:LetClause ;
          xqr:arg1  _:b13 ;
          xqr:arg2  _:b14 .
_:b13  a  xqr:Variable ;
          xqr:arg1  "t" .
_:b14  a  xqr:Path ;
          xqr:arg1  _:b6 ;
          xqr:arg2  "article" ;
          xqr:arg3  "title" .

### LET CLAUSE 3
_:b15  a  xqr:LetClause ;
          xqr:arg1  _:b16 ;
          xqr:arg2  _:b17 .
_:b16  a  xqr:Variable ;
          xqr:arg1  "j" .
_:b17  a  xqr:Path ;
          xqr:arg1  _:b6 ;
          xqr:arg2  "article" ;
          xqr:arg3  "journal" .
### LET CLAUSE 4
_:b18  a  xqr:LetClause ;
          xqr:arg1  _:b19 ;
          xqr:arg2  _:b20 .
_:b19  a  xqr:Variable ;
          xqr:arg1  "v" .
_:b20  a  xqr:Path ;
          xqr:arg1  _:b6 ;
          xqr:arg2  "article" ;
          xqr:arg3  "volume" .

### LET CLAUSE 5
_:b21 a  xqr:LetClause ;
          xqr:arg1  _:b22 ;
          xqr:arg2  _:b23 .
_:b22 a  xqr:Variable ;
          xqr:arg1  "y" .
_:b23  a  xqr:Path ;
          xqr:arg1  _:b6 ;
          xqr:arg2  "article" ;
          xqr:arg3  "year" .

### LET CLAUSE 6
_:b24 a  xqr:LetClause ;
          xqr:arg1  _:b25 ;
          xqr:arg2  _:b26 .
_:b25  a  xqr:Variable ;
          xqr:arg1  "ps" .
_:b26  a  xqr:Function ;
          xqr:functionName  "substring-before" ;
          xqr:arg1  _:b23 ;
          xqr:arg2  '-' .
_:b27  a  xqr:Path ;
          xqr:arg1  _:b6 ;
          xqr:arg2  "article" ;
          xqr:arg3  "pages" .
```

```
### LET CLAUSE 7
_:b28 a  xqr:LetClause ;
          xqr:arg1  _:b29 ;
          xqr:arg2  _:b30 .
_:b29 a  xqr:Variable ;
          xqr:arg1  "pe" .
_:b30  a  xqr:Function ;
          xqr:functionName  "substring-after" ;
          xqr:arg1  _:b23 ;
          xqr:arg2  '-' .

### RETURN CLAUSE
_:b26  a  xqr:ReturnClause ;
          xqr:arg1  _:b27 .
_:b27  a  xqr:Function ;
          xqr:functionName  "concat" ;
          xqr:arg1 '[]' ;

          xqr:arg2  _:b3 ;
          xqr:arg3  'bibo:authorList"' ;
          xqr:arg4  _:b9 ;
          xqr:arg5  '";';

          xqr:arg6  _:b3 ;
          xqr:arg7  'dcterms:title"' ;
          xqr:arg8  _:b13 ;
          xqr:arg9  '";';

          xqr:arg10  _:b3 ;
          xqr:arg11  'bibo:journal"' ;
          xqr:arg12  _:b16 ;
          xqr:arg13  '";';

          xqr:arg14  _:b3 ;
          xqr:arg15  'bibo:volume"' ;
          xqr:arg16  _:b19 ;
```

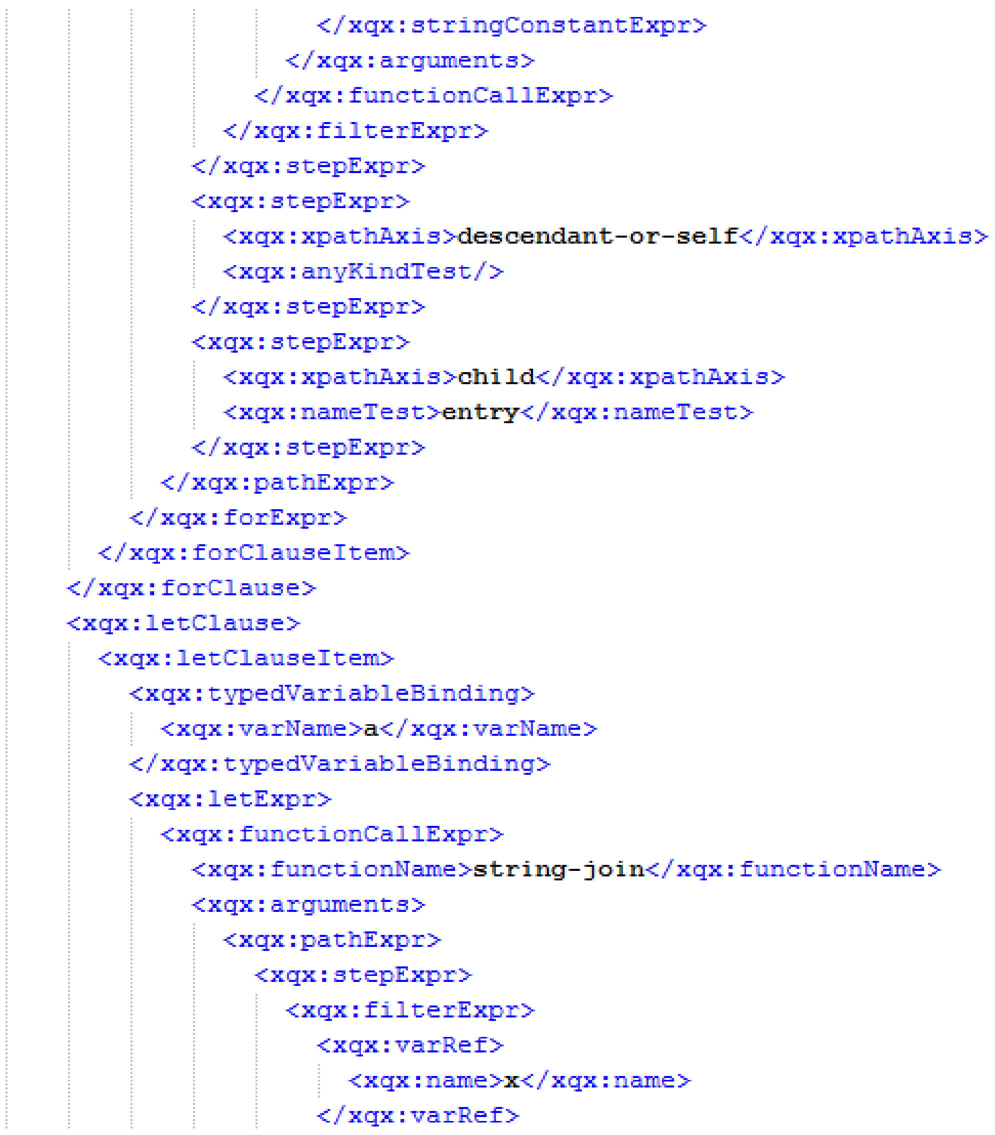
```
xqr:arg17  '";';

xqr:arg18  _:b3 ;
xqr:arg19  'dcterms:issued"' ;
xqr:arg20  _:b22;
xqr:arg21  '";';

xqr:arg22  _:b3 ;
xqr:arg23  'bibo:pageStart"' ;
xqr:arg24  _:b25 ;
xqr:arg25  '";';

xqr:arg26  _:b3 ;
xqr:arg27  'bibo:pageEnd"' ;
xqr:arg28 _:b29 ;
xqr:arg29 '";';
xqr:arg30 _:b3.
```

Appendix C: The result of mapping XQuery to XQueryX



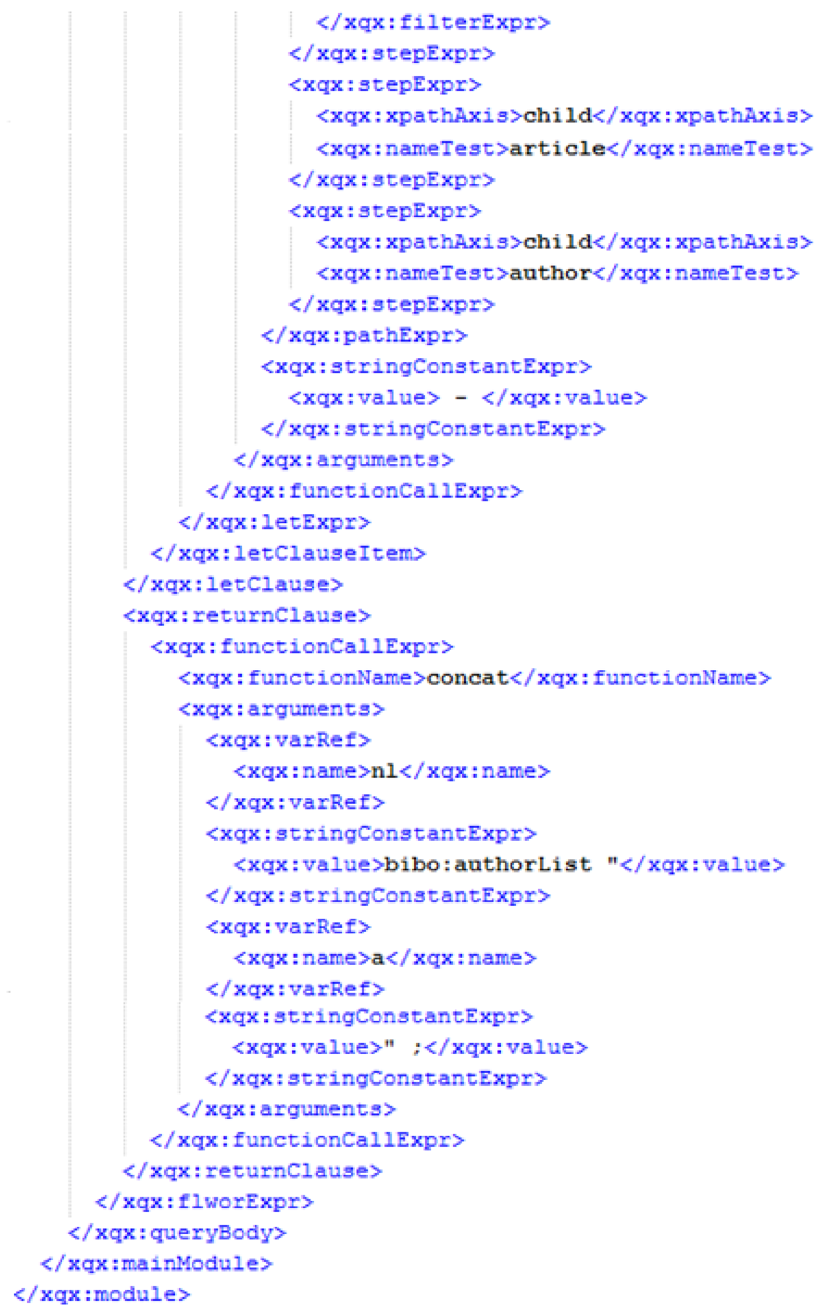
```
<?xml version="1.0"?>
<xqx:module xmlns:xqx="http://www.w3.org/2005/XQueryX"
            xmlns:xsi="http://www.w3.org/2001/XMLSchema-instance"
            xsi:schemaLocation="http://www.w3.org/2005/XQueryX
                                http://www.w3.org/2005/XQueryX/xqueryx.xsd"
  <xqx:mainModule>
    <xqx:prolog>
      <xqx:varDecl>
        <xqx:varName>nl</xqx:varName>
        <xqx:varValue>
          <xqx:stringConstantExpr>
            <xqx:value>&#10;</xqx:value>
          </xqx:stringConstantExpr>
        </xqx:varValue>
      </xqx:varDecl>
    </xqx:prolog>
    <xqx:queryBody>
      <xqx:flworExpr>
        <xqx:forClause>
          <xqx:forClauseItem>
            <xqx:typedVariableBinding>
              <xqx:varName>x</xqx:varName>
            </xqx:typedVariableBinding>
            <xqx:forExpr>
              <xqx:pathExpr>
                <xqx:stepExpr>
                  <xqx:filterExpr>
                    <xqx:functionCallExpr>
                      <xqx:functionName>doc</xqx:functionName>
                      <xqx:arguments>
                        <xqx:stringConstantExpr>
                          <xqx:value>xdb.xml</xqx:value>
```

Figure 7.1 - The result of mapping XQuery to XQueryX -1



```
                <xqx:stringConstantExpr>
              </xqx:arguments>
            </xqx:functionCallExpr>
          </xqx:filterExpr>
        </xqx:stepExpr>
        <xqx:stepExpr>
          <xqx:xpathAxis>descendant-or-self</xqx:xpathAxis>
          <xqx:anyKindTest/>
        </xqx:stepExpr>
        <xqx:stepExpr>
          <xqx:xpathAxis>child</xqx:xpathAxis>
          <xqx:nameTest>entry</xqx:nameTest>
        </xqx:stepExpr>
      </xqx:pathExpr>
    </xqx:forExpr>
  </xqx:forClauseItem>
</xqx:forClause>
<xqx:letClause>
  <xqx:letClauseItem>
    <xqx:typedVariableBinding>
      <xqx:varName>a</xqx:varName>
    </xqx:typedVariableBinding>
    <xqx:letExpr>
      <xqx:functionCallExpr>
        <xqx:functionName>string-join</xqx:functionName>
        <xqx:arguments>
          <xqx:pathExpr>
            <xqx:stepExpr>
              <xqx:filterExpr>
                <xqx:varRef>
                  <xqx:name>x</xqx:name>
                </xqx:varRef>
```

Figure 7.2 - The result of mapping XQuery to XQueryX -2



```
                      </xqx:filterExpr>
                    </xqx:stepExpr>
                    <xqx:stepExpr>
                      <xqx:xpathAxis>child</xqx:xpathAxis>
                      <xqx:nameTest>article</xqx:nameTest>
                    </xqx:stepExpr>
                    <xqx:stepExpr>
                      <xqx:xpathAxis>child</xqx:xpathAxis>
                      <xqx:nameTest>author</xqx:nameTest>
                    </xqx:stepExpr>
                  </xqx:pathExpr>
                  <xqx:stringConstantExpr>
                    <xqx:value> - </xqx:value>
                  </xqx:stringConstantExpr>
                </xqx:arguments>
              </xqx:functionCallExpr>
            </xqx:letExpr>
          </xqx:letClauseItem>
        </xqx:letClause>
        <xqx:returnClause>
          <xqx:functionCallExpr>
            <xqx:functionName>concat</xqx:functionName>
            <xqx:arguments>
              <xqx:varRef>
                <xqx:name>nl</xqx:name>
              </xqx:varRef>
              <xqx:stringConstantExpr>
                <xqx:value>bibo:authorList "</xqx:value>
              </xqx:stringConstantExpr>
              <xqx:varRef>
                <xqx:name>a</xqx:name>
              </xqx:varRef>
              <xqx:stringConstantExpr>
                <xqx:value>" ;</xqx:value>
              </xqx:stringConstantExpr>
            </xqx:arguments>
          </xqx:functionCallExpr>
        </xqx:returnClause>
      </xqx:flworExpr>
    </xqx:queryBody>
  </xqx:mainModule>
</xqx:module>
```

Figure 7.3 - The result of mapping XQuery to XQueryX -3