Semantic Based Access over XML Data [*]

Nikos Bikakis, Nektarios Gioldasis, Chrisa Tsinaraki,
Stavros Christodoulakis

Technical University of Crete, Department of Electronic and Computer Engineering
Laboratory of Distributed Multimedia Information Systems & Applications (TUC/MUSIC)
University Campus, 73100, Kounoupidiana Chania, Greece
{nbikakis, nektarios, chrisa, stavros}@ced.tuc.gr

Abstract. The need for semantic processing of information and services has lead to the introduction of tools for the description and management of knowledge within organizations, such as RDF, OWL, and SPARQL. However, semantic applications may have to access data from diverse sources across the network. Thus, SPARQL queries may have to be submitted and evaluated against existing XML or relational databases, and the results transferred back to be assembled for further processing. In this paper we describe the *SPARQL2XQuery* framework, which translates the SPARQL queries to semantically equivalent XQuery queries for accessing XML databases from the Semantic Web environment.

Keywords: Semantic Web, XML Data, Information Integration, Interoperability, Query Translation, SPARQL, XQuery, SPARQL to XQuery translation/transformation, SPARQL2XQuery.

1 Introduction

XML has been extremely successful for information exchange in the Web. Over the years XML was established as a tool for describing the content of diverse structured or unstructured resources in a flexible manner. The information transferred with XML documents across the internet lead to needs of systematic management of the XML documents in organizations. XML Schema and XQuery [7] were developed to give the users database management functionality analogous to the Relational Model and SQL. In the Web application environment the XML Schema acts also as a wrapper to relational content that may coexist in the databases.

The need for semantic information processing in the Web on the other hand has lead to the development of a different set of standards including OWL, RDF and SPARQL[5]. Semantic Web application developers expect to utilize SPARQL for accessing RDF data. However, information across the network may be managed by databases that are based on other data models such as XML Schema or the Relational model. Converting all the data that exist in the XML databases into Semantic Web data is unrealistic due to the different data models used (and enforced by different

[*] An extended version of this paper is available at [20].

standardization bodies), the management requirements (including updates), the difficulties in enforcing the original data semantics, ownership issues, and the large volumes of data involved

In this paper we propose an environment where Semantic Web users write their queries in SPARQL, and appropriate interoperability software undertakes the responsibility to translate the SPARQL queries into semantically equivalent XQuery queries in order to access XML databases across the net. The results come back as RDF (N3 or XML/RDF) or XML [1] data. This environment accepts as input a set of mappings between an OWL ontology and an XML Schema. We support a set of language level correspondences (rules) for mappings between RDFS/OWL and XML Schema. Based on these mappings our framework is able to translate SPARQL queries into semantically equivalent XQuery expressions as well as to convert XML Data in the RDF format. Our approach provides an important component of any Semantic Web middleware, which enables transparent access to existing XML databases.

The framework has been smoothly integrated with the *XS2OWL* framework [15], thus achieving not only the automatic generation of mappings between XML Schemas and OWL ontologies, but also the transformation of XML documents in RDF format.

The design objectives for the development of the *SPARQL2XQuery* framework have been the following: a) Capability of translating every query compliant to the SPARQL grammar b) Strict compliance with the SPARQL semantics, c) Independence from query engines and working environments for XQuery, d) Production of the simplest possible XQuery expressions, e) Construction of XQuery expressions so that their correspondence to SPARQL can be easily understood, f) Construction of XQuery expressions that produce results that do not need any further processing, and g) In combination with the previous objectives, construction of the most efficient XQuery expressions possible.

The rest of the paper is organized as follows: In Section 2 the related work is presented. The mappings used for the translation as well as their encoding are described in Section 3. Section 4 describes the query translation process. An example presented at Section 5. The transformation of the query results described at Section 6. The paper concludes in section 7.

2 Related Work

Various attempts have been made in the literature to address the issue of accessing XML data from within Semantic Web Environments [2, 3, 6, 8, 9, 10, 11, 15, 16, 17, 18]. More relevant to our work are those that use SPARQL as a manipulation language. To this end, the *SAWSDL Working Group* [8] uses XSLT to convert XML data into RDF and a combination of SPARQL and XSLT for the inverse. Other approaches [9, 10, 11] combine Semantic Web and XML technologies to provide a bridge between XML and RDF environments. *XSPARQL* [11] combines SPARQL and XQuery in order to achieve Lifting and Lowering. In the Lifting scenario (which is relevant to our work), *XSPARQL* uses XQuery expressions to access XML data and SPARQL Construct queries for converting the accessed data into RDF. The main drawback of these approaches is that there is no automatic way to express an XML retrieval query in SPARQL. Instead, the user must be aware of the XML Schema and create his/her information retrieval query accordingly (XQuery or XSLT). In our work, the user is not expected to know the underlying XML Schema; (s)he expresses his/her query only in SPARQL in terms of the knowledge that (s)he is aware of, and (s)he is able to retrieve data that exist in XML databases. The aforementioned attempts, as well as others [12, 13, 14] that try to bridge relational databases with the Semantic Web using SPARQL, show that the issue of accessing legacy data sources from within Semantic Web environments is a valuable and challenging one.

3 Mapping OWL to XML Schema

The framework described here allows XML encoded data to be accessed from Semantic Web applications that are aware of some ontology encoded in OWL. To do that, appropriate mappings between the OWL ontology (*O*) and the XML Schema (*XS*) should exist. These mappings may be produced either automatically, based on our previous work in the *XS2OWL* framework [15], or manually through some mapping process carried out by a domain expert. However, the definition of mappings between OWL ontologies and XML Schemas is not the subject of this paper. Thus, we do not focus on the semantic correctness of the defined mappings. We neither consider what the mapping process is, nor how these mappings have been produced

Such a mapping process has to be guided from language level correspondences. That is, the valid correspondences between the OWL and XML Schema language constructs have to be defined in advance. The language level correspondences that have been adopted in this paper are well-accepted in a wide range of data integration approaches [2, 3, 6, 15, 16, 17]. In particular, we support mappings that obey the following language level correspondence rules: *O* Class corresponds to *XS* Complex Type, *O* DataType Property corresponds to *XS* Simple Element or Attribute, and *O* Object Property corresponds to *XS* Complex Element.

Then, at the schema level, mappings between concrete domain conceptualizations have to be defined (e.g. the *employee* class is mapped to the *worker* complex type) either manually, or automatically, following the correspondences established at the language level.

At the schema level mappings a mapping relationship between *O* and an *XS* is a binary association representing a semantic association among them. It is possible that for a single ontology construct more than one mapping relationships are defined. That is, a single source ontology construct can be mapped to more than one target XML Schema elements (1:n mapping) and vice versa, while more complex mapping relationships can be supported.

3.1 Encoding of the Schema Level Mappings

Since we want to translate SPARQL queries into semantically equivalent XQuery expressions that can be evaluated over XML data following a given (mapped) schema, we are interested in XML data representations. As a consequence, based on schema level mappings for each mapped ontology class or property, we store a set of XPath expressions (*"XPath set"* for the rest of this paper) that address all the corresponding instances (XML nodes) in the XML data level. In particular, based on the schema level mappings, we construct:

- A **Class XPath Set** X_C for each mapped class *C*, containing all the possible XPaths of the complex types to which the class *C* has been mapped to.
- A **Property XPath Set** X_{Pr} for each mapped property *Pr*, containing all the possible XPaths of the elements or/and attributes to which *Pr* has been mapped.

Example 1: Encodings of Mappings

Fig. 1 shows the mappings between an OWL Ontology and an XML Schema.

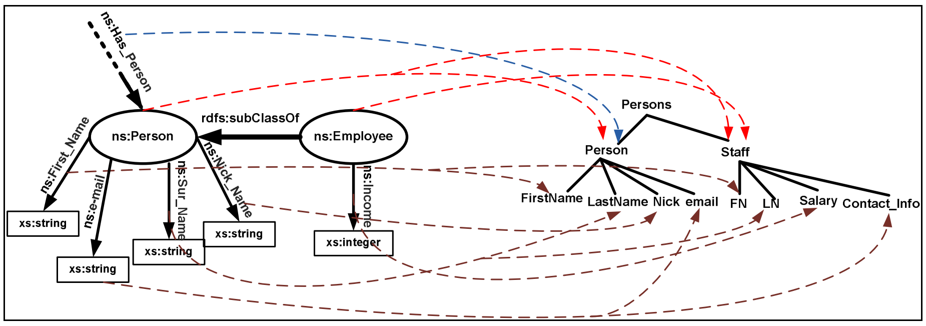

Fig. 1. Mappings Between OWL & XML

To better explain the defined mappings, Fig. 1 shows the structure that the XML documents (which follow this schema) will have. The encoding of these mappings in our framework is shown in Fig. 2.

Classes:	**DataType Properties:**
$\mathbf{X}_{ns:Person}$={/Persons/Person, /Persons/Staff}	$\mathbf{X}_{ns:First\_Name}$={/Persons/Person/FirstName, /Persons/Staff/FN}
$\mathbf{X}_{ns:Employee}$={/Persons/Staff}	$\mathbf{X}_{ns:Sur\_Name}$={/Persons/Person/LastName, /Persons/Staff/LN}
	$\mathbf{X}_{ns:Nick\_Name}$={/Persons/Person/Nick }
Object Properties:	$\mathbf{X}_{ns:e\text{-}mail}$={/Persons/Person/email, /Persons/Staff/Contact_Info}
$\mathbf{X}_{ns:Has\_Person}$={/Persons/Person }	$\mathbf{X}_{ns:Income}$={/Persons/Staff/Salary}

Fig. 2. Mappings Encodings

4 Query Translation Process

In this section we present in brief the entire translation process using a UML activity diagram Fig. 3 shows the entire process which starts taking as input the given SPARQL query and the defined mappings between the ontology and the XML Schema (encoded as described in the previous sections). The query translation process comprises the activities outlined in the following paragraphs.

4.1 SPARQL Graph Pattern Normalization

The *SPARQL Graph Pattern Normalization* activity re-writes the Graph-Pattern (*GP*) of the SPARQL query in an equivalent normal form based on equivalence rules. The SPARQL *GP* normalization is based on the *GP* expression equivalences proved in [4] and re-writing techniques. In particular, each *GP* can be transformed in a sequence *P1 UNION P2 UNION P3 UNION…UNION Pn,* where *Pi* ($1 \le i \le n$) is a Union-Free *GP* (i.e. *GPs* that do not contain Union operators) [4]. This makes the *GP* translation process simpler and more efficient, since it decomposes the entire query pattern into sub-patterns that can be processed independently of each other.

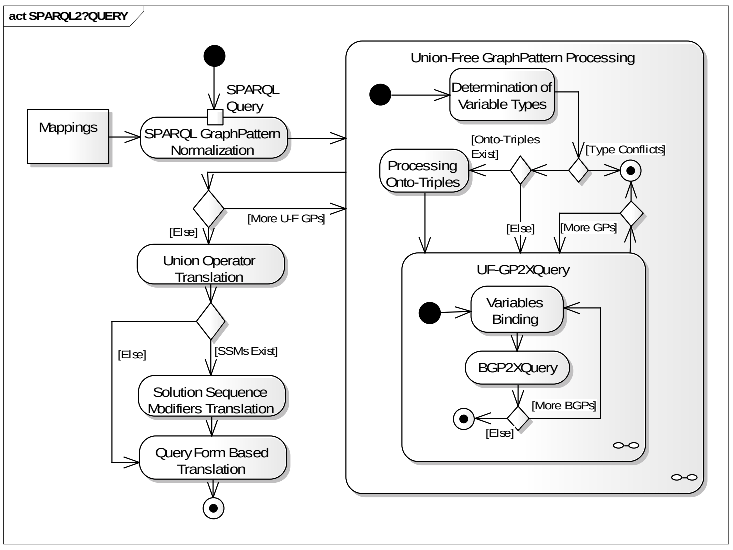

Fig. 3 Overview of the SPARQL Translation Process

4.2 Union-Free Graph Pattern (UF-GP) Processing

The *UF-GP Processing* translates the constituent *UF-GPs* into semantically equivalent XQuery expressions. The *UF-GP Processing* activity is a composite one, with various sub-activities. This is actually the step that most of the “real work” is done since at this step most of the translation process takes place. The *UF-GP Processing* activity is decomposed in the following sub-activities:

Determination of Variable Types. This activity examines the type of each variable referenced in each UF-GP in order to determine the form of the results and, consequently, the syntax of the Return clause in XQuery. Moreover, variable types are used by the *“Processing Onto-Triples”* and *“Variables Bindings”* activities. Finally, this activity performs consistency checking in variable usage in order to detect any possible conflict (e.g. the same variable name is used in the definitions of variables of different types in the same *UF-GP*). In such a case, the *UF-GP* is not going to be translated, because it is not possible to be matched with any RDF dataset.

We define the following variable types: The *Class Instance Variable Type* (*CIVT*), The *Literal Variable Type* (*LVT*), The *Unknown Variable Type* (*UVT*), The *Data Type Predicate Variable Type* (*DTPVT*), The *Object Predicate Variable Type* (*OPVT*), The *Unknown Predicate Variable Type* (*UPVT*).

The form of the results depends on the variable types and they are structured in such a way that allows their transformation to RDF syntax. The transformation can be done by processing the information regarding the form of the results and the input mappings. In order to allow the construction of result forms, appropriate XQuery functions (using standard XQuery expressions) have been implemented (like *func:CIVT*, etc.).

Processing Onto-Triples. *Onto-Triples* actually refer to the ontology structure and/or semantics. The main objective of this activity is to process onto-triples against the ontology (using SPARQL) and based on this analysis to bind (i.e. assigning the relevant XPaths to variables) the correct XPaths to variables contained in the onto-triples. These bindings are going to be used in the next steps as input *Variable Binding* activity. This activity processes *Onto-Triples* using standard SPARQL in order to perform any required inference so that any schema-level query semantics to be analyzed and taken into account later on in the translation process. Since we are using SPARQL for *Onto-Triple* processing against the ontology, we can process any given *Onto-Triple* regardless the complexity of its matching against the ontology graph.

UF-GP2XQuery. This activity translates the *UF-GP* into semantically equivalent XQuery expressions. The concept of a *GP*, and thus the concept of *UF-GF*, is defined recursively. The *BGP2XQuery* activity translates the basic components of a *GP* (i.e. Basic Graph Patterns-*BGPs* which are sequences of triple patterns and filters) into semantically equivalent XQuery expressions. To do that a variables binding step is needed. Finally, *BGPs* in the context of a *GP* have to be properly associated. That is, to apply the SPARQL operators among them using XQuery expressions and functions. These operators are: *OPT*, *AND*, and *FILTER* and are implemented using standard XQuery expressions without any ad hoc processing.

- **Variables Binding.** In the translation process the term "variable bindings" is used to describe the assignment of the correct XPaths to the variables referenced in a given Basic Graph Pattern (*BGP*), thus enabling the translation of *BGP* to XQuery expressions. In this activity, *Onto-Triples* are not taken into account since their processing has taken place in the previous step and their bindings are used as input in this activity. The same holds for Filters, since they don't affect the binding process (more details can be found at [19]).

- **BGP2XQuery.** This activity translates the *BGPs* to semantically equivalent XQuery expressions based on the *BGP2XQuery* algorithm. The algorithm manipulates a sequence of triple patterns and filters (i.e. a *BGP*) and translates them into XQuery expressions, thus allowing the evaluation of a *BGP* on a set of XML data. The algorithm takes as input the mappings between the ontology and the XML schema, the *BGP*, the determined variable types, as well as the variable bindings and generates XQuery expressions (more details can be found at [19]).

4.3 Union Operator Translation

This activity translates the *UNION* operator that appears among *UF-GPs* in a *GP*, by using the *Let* and *Return* XQuery clauses in order to return the union of the solution sequence produced by the *UF-GPs* to which the Union operator applies.

4.4 Solution Sequence Modifiers Translation

This activity translates the SPARQL solution sequence modifiers. Solution Modifiers are applied on a solution sequence in order to create another, user desired, sequence. The modifiers supported by SPARQL are *Distinct*, Order By, *Reduced*, *Limit* and *Offset*.

For the implementation of the *Distinct* and *Reduced* modifiers, our software generates XQuery functions (in standard XQuery syntax) (*func:DISTINCT*, *func:REDUCED*) according to the number and the names of the variables for which the duplicate elimination is to be performed. Regarding the rest of the solution sequence modifiers, the next table shows the XQuery expressions and built-in functions that are used for their translation in XQuery (the XQuery variable *$Results* has been bound to the solution sequence produced by XQuery expressions, and *N, M* are positive integers).

Table 1. Translation of Solutions Sequence Modifiers

SPARQL Modifier	*XQuery Translation*
LIMIT N	**return**($Results[**position**( )<= N])
OFFSET N	**return**($Results[**position**( )> N])
LIMIT N & OFFSET M	**return**($Results[**position**( )> M **and position**( )<= N+M])

ORDER BY DESC(?x) **ASC**(?y)	**for** $res **in** $Results
	order by $res/x **descending empty least**, $res/y **empty least**
	return $res

4.5 Query Forms Based Translation

SPARQL has four forms of queries (*Select, Ask, Construct* and *Describe*). According to the query form, the structure of the final result is different. The query translation is heavily dependent on the query form. In particular, after the translation of any solution modifier is done, the generated XQuery is enhanced with appropriate expressions in order to achieve the desired structure of the results (e.g. to construct an RDF graph, or a result set) according to query form.

5 Example

We demonstrate in this example the use of the described framework in order to allow a SPARQL query to be evaluated in XML Data (based on Example 1). Fig. 4 shows how a given SPARQL query is translated by our framework into a semantically equivalent XQuery query.

Consider the query: *"For Person subclasses, return their instances, their last name and their income, the first name of which is "Nick", the last name begins with "B", and they have an e-mail address. The (existence of) income is optional. The query will return at most 20(LIMIT 20) solutions ordered by last name value at descending order and skipping the first 5 solutions (OFFSET 5)*".

SPARQL Query:

```
PREFIX ns:    <http://example.com/ns#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
PREFIX rdf:   <http://www.w3.org/1999/02/22-rdf-syntax-ns#>

SELECT  ?empl ?lname ?inc
WHERE{ { ?emplCl  rdfs:subClassOf  ns:Person .      ⎫
         ?empl      rdf:type            ? emplCl .    ⎪
         ?empl   ns:First_Name       "Nick" .         ⎬ BGP_1
         ?empl   ns:Sur_Name         ?lname .         ⎪
         ?empl   ns:email            ?email .         ⎪
         FILTER regex( ?lname, "^B")      }           ⎭
       OPTIONAL{ ?empl  ns:Income ?inc. }            } BGP_2
     }
ORDER BY DESC(?lname)
LIMIT 20  OFFSET 5
```

Translated XQuery Query :

```
declare namespace func = "http://www.music.tuc.gr/funcs";
let $doc := collection("http://www.music.tuc.gr/...")
let $Modified_Results :=(
   let $Results :=(
      let $BGP_1:=(
         for $empl   in  $doc/Persons/Staff[./FN= "Nick"]
         for $lname in  $empl/LN
         let  $email :=  $empl/Contact_Info
         where( exists($email) and matches($lname, "^B" )  )
         return(<Result> <empl>{func:CIVT($empl)}</empl>,
                            <lname>{ string($lname)}</lname> </Result>)
      )
      let $BGP_2:=(
         for $empl in $doc/Persons/Staff
         for $inc in $empl/Salary
         return(<Result> <empl>{func:CIVT($empl)}</empl>,
                            <inc>{ string($inc)}</inc> </Result>)
      )
      return ( func:OPTIONAL($BGP_1, $BGP_2) )
   )
   return (let $Ordered_Results:=(
                             for $iter in $Results
                             order by $iter/lname descending empty least
                             return($iter) )
         return ($Ordered_Results[position( )>5 and position( )<=25]))
)
return (<Results>{$Modified_Results }</Results>)
```

Fig. 4. SPARQL Query Translation Example

6 Transformation of the Query Results

An important issue in the entire approach is the structure of the returned results. In our work and for the *Ask* and *Select* query forms we encode the returned results according to the SPARQL Query Result XML Format [1], which is a W3C recommendation. Moreover the values returned with the results, can be easily transformed into RDF (N3 or RDF/XML) syntax by processing the information of the results and the input mappings.

7 Conclusions

We have presented an environment that allows the evaluation of SPARQL queries over XML data which are stored in XML databases and accessed with the XQuery language. The environment assumes that a set of mappings between the OWL ontology and the XML Schema exists. The mappings obey certain well accepted language correspondences.

The *SPARQL2XQuery* framework has been implemented as a prototype software service using Java related technologies (Java 2SE, Axis2, and Jena) on top of the Berkeley DB XML. The service can be configured with the appropriate mappings (between an ontology and an XML Schema) and translates the input SPARQL queries into XQuery queries that are answered over the XML Database.

This work is part of as more generic framework that we are pursuing which aims to providing algorithms, proofs and middleware for the transparent access from the Semantic Web environment to federated heterogeneous databases across the web.

8 References

1. Beckett D. (eds), "SPARQL Query Results XML Format". W3C Recommendation, 15 January 2008, (http://www.w3.org/TR/rdf-sparql-XMLres/).
2. Bohring H., Auer S.: "Mapping XML to OWL Ontologies". Leipziger Informatik-Tage 2005: 147-156
3. Lehti P., Fankhauser P.: "XML Data Integration with OWL: Experiences & Challenges", Proceedings of the 2004 International Symposium on Applications and the Internet
4. J. Perez, M. Arenas, C. Gutierrez. Semantics and Complexity of SPARQL. 5th International Semantic Web Conference (ISWC-06), November 2006.
5. Prud'hommeaux E., Seaborne A. (eds), "SPARQL Query Language for RDF". W3C Recommendation, 15 January 2008. (http://www.w3.org/TR/rdf-sparql-query/).
6. Rodrigues T., Rosa P, Cardoso J., "Mapping XML to Exiting OWL ontologies", International Conference WWW/Internet 2006, Murcia, Spain, 5-8 October 2006.
7. Siméon J., Chamberlin D. (eds): XQuery 1.0: an XML Query Language. W3C Recommendation, 23 Jan. 2007. http://www.w3.org/TR/xquery/.
8. Joel Farrell and Holger Lausen. Semantic Annotations for WSDL and XML Schema. W3C Recommendation, W3C, August 2007. Available at http://www.w3.org/TR/sawsdl/
9. Sven Groppe, Jinghua Groppe, Volker Linnemann, Dirk Kukulenz, Nils Hoeller, Christoph Reinke: Embedding SPARQL into XQuery/XSLT. SAC 2008: 2271-2278
10. Matthias Droop, Markus Flarer, Jinghua Groppe, Sven Groppe, Volker Linnemann, Jakob Pinggera, Florian Santner, Michael Schier, Felix Schoepf, Hannes Staffler, Stefan Zugal: "Embedding XPATH Queries into SPARQL Queries" In Proc. of the 10th International Conference on Enterprise Information Systems(ICEIS 2008)
11. Waseem Akhtar, Jacek Kopecký, Thomas Krennwallner, Axel Polleres : XSPARQL: Traveling between the XML and RDF Worlds - and Avoiding the XSLT Pilgrimage. ESWC 2008:432-447
12. Christian Bizer, Richard Cyganiak : D2R Server. http://www4.wiwiss.fu-berlin.de/bizer/d2r-server/index.html
13. OpenLink Software : Virtuoso Universal Server. http://virtuoso.openlinksw.com/
14. CCNT Lab. Zhejiang Univ. China : Dart Grid. http://ccnt.zju.edu.cn/projects/dartgrid/
15. Tsinaraki C., Christodoulakis S., "Interoperability of XML Schema Applications with OWL Domain Knowledge and Semantic Web Tools". In Proc. of the ODBASE 2007.

16. Cruz I.R., Huiyong Xiao Feihong Hsu: “An Ontology-based Framework for XML Semantic Integration”, Database Engineering and Applications Symposium, 2004.
17. V.Christophides, G. Karvounarakis, I. Koffina, G. Kokkinidis, A. Magkanaraki, D. Plexousakis, G. Serfiotis, V. Tannen: “The ICS-FORTH SWIM: A Powerful Semantic Web Integration Middleware”. Proceedings of the First International Workshop on Semantic Web and Databases 2003 (SWDB 2003), pages 381-393.
18. Bernd Amann, Catriel Beeri, Irini Fundulaki, Michel Scholl: Querying XML Sources Using an Ontology-Based Mediator. CoopIS/DOA/ODBASE 2002: 429-448
19. Bikakis N., Gioldasis N., Tsinaraki C., Christodoulakis S.: “Querying XML Data with SPARQL” In Proceeding. of the 20th International Conference on Database and Expert Systems Applications (DEXA'09)
20. Bikakis N., Gioldasis N., Tsinaraki C., Christodoulakis S.: “The SPARQL2XQuery Framework” Technical Report *http://www.music.tuc.gr/reports/SPARQL2XQUERY.PDF*