Optimising XML–RDF Data Integration
A Formal Approach to Improve XSPARQL Efficiency

Stefan Bischof

Siemens AG Österreich, Siemensstraße 90, 1210 Vienna, Austria
bischof.stefan@siemens.com

Abstract The Semantic Web provides a wealth of open data in RDF format. XML remains a widespread format for data exchange. When combining data of these two formats several problems arise due to representational incompatibilities. The query language XSPARQL, which is built by combining XQuery and SPARQL, addresses some of these problems. However the evaluation of complex XSPARQL queries by a naive implementation shows slow response times. Establishing an integrated formal model for a core fragment of XSPARQL will allow us to improve performance of query answering by defining query equivalences.

Keywords: Data Integration, Query Optimisation, XQuery, SPARQL

1 Integrating XML and RDF Data Lacks Efficiency

Data exchange became faster and more convenient by using the Internet. Two of the predominant formats to exchange data over the Internet but also inside organisations are *XML*, used for business data, financial data, etc., and *RDF*, the data format of the Semantic Web. While XML is tree based with relevant child order, RDF data is unordered due to its graph structure. The W3C recommended query languages for XML and RDF are XQuery and SPARQL, respectively. A brief example scenario: Fred has to send a report containing the number of bought items for each customer to Julie. Julie expects a XML document while Fred stores his data in a RDF triple store. Fred also has to integrate XML documents from Julie into his triple store. When using standard query languages Fred ends up with several queries in different languages glued together by scripts.

XSPARQL [3] is a combined query language making transformation and integration of XML and RDF data easier. XSPARQL extends XQuery by two additional grammar expressions: the *SparqlForClause* to use SPARQL's graph pattern matching facility including operators, and the *ConstructClause* to allow straightforward creation of RDF graphs. Listing 1 shows an example query. Fred produces the XML document for Julie by taking the RDF graph as input. Graph patterns are used to query the RDF graph and the variables of the graph pattern are then used by any XQuery expression. Since XSPARQL is based on XQuery, a Turing-complete language, it supports all kinds of more sophisticated data manipulations. Fred can also use XSPARQL to convert Julie's XML documents to custom RDF by using a single query containing a `construct` template.

Listing 1. XSPARQL query: List customers with number of bought items

```
for * where { [] foaf:name $name ; :id $id . }
return <customer name="{$name}"> { count(
  for * where { [] :buyer [:id $id]; :itemRef []. }
  return <item/> ) } </customer>
```

Efficiency is important when transforming data between XML and RDF. Evaluating complex XSPARQL queries, i.e., queries containing nested graph patterns, shows slow query response times with the current prototype.[1] The performance impact can be explained by the architecture which rewrites XSPARQL queries to XQuery queries containing external calls to a SPARQL engine. The main advantages of such an implementation are reuse of state of the art query optimisation as well as access to standard XML databases and triple stores. But for complex XSPARQL queries a high number of SPARQL queries (similar to each other) are yielded, resulting in a major performance impact [3–5]. Listing 1 contains a nested *SparqlForClause* in line 3, which depends on the specific customer ID (`$id`). The implementation issues a separate SPARQL query for each customer. Simple join reordering [3, 4] improves query answering performance. But there is still a performance gap between simple flat queries and complex nested queries, thus optimisations on a more fundamental level are needed.

The XQuery semantics specification [8] formalism, i.e., natural semantics, is not well suited for concisely expressing query equivalences. As opposed to Relational Algebra, which serves as the basis for query languages like SPARQL, natural semantics uses calculus-like rules to specify type inference and evaluation semantics. Since the XSPARQL semantics reuses the formalism of XQuery, a concise description of possible optimisations is inhibited by the formalism.

To find and express new optimisations and prove their correctness, we need a more suitable formalism. Finding such a formalism is the first goal of the presented PhD topic. Like other formalisms used for query optimisation we also use only a core fragment of the query language to optimise. We thus propose an integrated formal model of an XSPARQL core fragment called XS.

Section 2 gives an overview of related work and describes open problems. Section 3 explains the approach we propose which involves creating a core formalism to express different kinds of optimisations by query equivalence and rules for cost-based optimisations. Lastly Sect. 4 outlines the research methodology for addressing the efficiency problem by a formal approach.

2 Related Work

Some published approaches to combine XML and RDF data use either XML or RDF query languages or a combination. But none of these approaches is advanced enough to address "cross-format" optimisation of such transformations. In general such optimisations could be implemented by translating queries completely to a

[1] An online demo and the source code are available at `http://xsparql.deri.org/`

single existing query language, such as XQuery, and shift optimisation and query evaluation to an XQuery engine. Another approach to implement a combined query language is to build an integrated evaluation engine from scratch.

Translate SPARQL to XQuery. Groppe et al. [12] present a language extension syntactically similar to XSPARQL, which extends XQuery/XSLT to allow SPARQL queries as nested expressions. After an initial RDF to XML data translation the query translation uses an intermediary algebra allowing SPARQL optimisation, resulting in a pure XQuery/XSLT query. Fischer et al. [9] implement a similar approach of rewriting SPARQL queries to XQuery. By using a more sophisticated initial XML data transformation and heavier triple pattern join reordering, they achieve better performance for queries with simple joins and filters. Other approaches relying on ontology information are not relevant for the current work, as a query language like XSPARQL gives the user fine grained control over the output and intermediary transformations, but does not work automatically. None of the above approaches address "cross-format" optimisation. As we have found in our initial practical evaluation, query engines not catering for the combination of both languages will suffer a severe performance impact when evaluating complex queries. Complex queries containing nested graph patterns will occur frequently in practice when transforming data from RDF to XML because of the inherent tree shape of the target XML documents.

Optimisation Formalisation. The current formal specification of XSPARQL, based on XQuery, is too verbose to allow comprehensible specification of optimisations. Therefore we are looking for a formalisation supporting our specific requirements: concise semantics and optimisation specification of a language fragment.

XQuery optimisation is an obvious starting point when investigating optimisation of XSPARQL. Some of the recent works on XQuery optimisation use custom algebras [1,13,14]. Other approaches [2,6,7,10,11] map XQuery to standard logics (Datalog/logic programming, monadic second order logic) in order to profit from well studied properties. A native algebra easily transfers to an implementation prototype, however it makes comparisons to other formalisms hardly feasible. Using a well known formalisation like Datalog would make custom XSPARQL optimisations easier since existing optimisation approaches could be reused.

3 A Formal Model for Cross-Format Optimisation

The first step in addressing the XSPARQL efficiency problem, is to express the query language semantics in a formal way. Since the XQuery semantics formalism is operational and rather verbose, we look into alternative formalisms capable of expressing the language semantics, at least for a core fragment, as well as expression equivalences and corresponding proofs.

We aim to formalise an XSPARQL core fragment called *XS*, which is expressive enough to cover relevant queries or query parts, and simple enough to be able to use query equivalences for optimisation heuristics. XS will therefore be a well behaving formal model providing a unified representation and manipulation framework for (unordered) graph data and (ordered) tree data at the same time. The optimisation heuristics can also be used for related approaches since the model works on the data models of XML and RDF. A cost model will be created, enabling cost based optimisations as well. We will gather data about the underlying data distribution for the cost model. We will build an XS prototype to allow comparing the query answering performance to the naive implementation.

Optimisation of query languages operating on the data models of XML and RDF at the same time are not well studied so far. Since SPARQL and XQuery are based on different paradigms—SPARQL is a declarative language comparable to SQL, while XQuery is a functional query language—formalisations differ greatly, even when considering only fragments, as usually studied for optimisation.

One Limitation of the approach is expressivity: although XSPARQL is very expressive, we aim at a concise formalisation. Thus XS will disallow many queries. Even though we are aiming at finding a good compromise between optimisation and expressivity, increasing expressivity is out of scope of this work.

4 Research Methodology

Finding novel optimisations for querying across formats requires several tasks: literature search, formalisation, theoretical verification, prototyping, practical evaluation and implementation of a relevant use case. All of these tasks are not executed strictly in sequence, but rather in several iterative refinement steps.

Literature Search. For clearly defining the research problem and ensuring novelty of the approach, we are currently performing an extensive literature search. Data integration and data exchange are recent topics attracting researchers from different domains. Integrating data from different representations however, has not seen broad attention. A second topic currently under investigation is finding an appropriate formalisation for XS.

Core Fragment Isolation and Optimisation. We will isolate an XSPARQL core fragment which allows expressing practically relevant queries and holds optimisation potential. Optimisations are defined by query equivalences.

Theoretical Evaluation. The core fragment allows theoretical correctness verification. Using XS we will prove theoretical properties such as complexity bounds and query/expression equivalences.

Prototype. A prototype implementation is needed to devise practical evaluations and to implement a demonstration use case. The prototype will be built using state of the art technology but should still be kept simple enough to allow quick adaption. We also plan to publish concrete use case solutions to show feasibility and relevance of our approach in practise.

Practical Evaluation. In previous work we proposed the benchmark suite XMarkRDF [4] to measure the performance of RDF to XML data transformation. XMarkRDF is derived from the XQuery benchmark suite XMark. We will extend XMarkRDF by queries covering specific types of joins as addressed by the XS optimisations. With this benchmark suite we plan to compare our optimisations to the current implementation and to comparable implementations such as the translator presented by Groppe et al. [12].

In summary the research topic includes isolating an XSPARQL core fragment, finding or creating a suitable formalisation (a unified graph-tree data processing framework), describing optimisations by query equivalences, proving soundness, evaluating performance practically and showing applicability in a prototype. With the unified formal model for XML–RDF querying, we aim to provide a tool to formally study heterogeneous data integration and improve query performance.

Acknowledgements. The work of Stefan Bischof will be partly funded by a PhD thesis grant from Siemens AG Österreich. The goal of this grant is that in the course of his thesis, he will investigate the potential to deploy the developed technologies within real-life data integration use cases within Siemens. Preliminary results have been partly funded by Science Foundation Ireland grant no. SFI/08/CE/I1380 (Lion-2) and an IRCSET grant.

References

1. Beeri, C., Tzaban, Y.: SAL: An Algebra for Semistructured Data and XML. In: WebDB (Informal Proceedings) 1999. pp. 37–42 (June 1999)
2. Benedikt, M., Koch, C.: From XQuery to Relational Logics. ACM Trans. Database Syst. 34(4), 25:1–25:48 (December 2009)
3. Bischof, S., Decker, S., Krennwallner, T., Lopes, N., Polleres, A.: Mapping between RDF and XML with XSPARQL. Tech. rep., DERI (Mar 2011), `http://www.deri.ie/fileadmin/documents/DERI-TR-2011-04-04.pdf`
4. Bischof, S., Decker, S., Krennwallner, T., Lopes, N., Polleres, A.: Mapping between RDF and XML with XSPARQL (2011), under Submission
5. Bischof, S., Lopes, N., Polleres, A.: Improve Efficiency of Mapping Data between XML and RDF with XSPARQL. In: Rudolph, S., Gutierrez, C. (eds.) RR 2011. LNCS, vol. 6902, pp. 232–237. Springer (August 2011)
6. ten Cate, B., Lutz, C.: The Complexity of Query Containment in Expressive Fragments of XPath 2.0. J. ACM 56(6), 31:1–31:48 (September 2009)
7. ten Cate, B., Marx, M.: Axiomatizing the Logical Core of XPath 2.0. Theor. Comp. Sys. 44(4), 561–589 (April 2009)
8. Draper, D., Fankhauser, P., Fernández, M., Malhotra, A., Rose, K., Rys, M., Siméon, J., Wadler, P.: XQuery 1.0 and XPath 2.0 Formal Semantics (2nd Ed.). W3C Recommendation, `http://www.w3.org/TR/2010/REC-xquery-semantics-20101214/`
9. Fischer, P.M., Florescu, D., Kaufmann, M., Kossmann, D.: Translating SPARQL and SQL to XQuery. In: XML Prague 2011. pp. 81–98 (March 2011)
10. Gottlob, G., Koch, C., Pichler, R.: Efficient Algorithms for Processing XPath Queries. ACM Trans. Database Syst. 30, 444–491 (June 2005)
11. Gottlob, G., Koch, C., Pichler, R., Segoufin, L.: The Complexity of XPath Query Evaluation and XML Typing. J. ACM 52(2), 284–335 (March 2005)
12. Groppe, S., Groppe, J., Linnemann, V., Kukulenz, D., Hoeller, N., Reinke, C.: Embedding SPARQL into XQuery/XSLT. In: SAC'08. pp. 2271–2278. ACM (2008)
13. Jagadish, H.V., Lakshmanan, L.V., Srivastava, D., Thompson, K.: TAX: A Tree Algebra for XML. In: DBPL 2001. LNCS, vol. 2397, pp. 149–164. Springer (2002)
14. Zhang, X., Pielech, B., Rundesnteiner, E.A.: Honey, I Shrunk the XQuery!: an XML Algebra Optimization Approach. In: WIDM '02. pp. 15–22. ACM (2002)