Unrolling SPARQL in SPARQL: Exploring SPARQL views as a declarative rewriting mechanism

Jitse De Smet[1], Maxime Jakubowski[2] and Ruben Taelman[1]

[1]*IDLab, Department of Electronics and Information Systems, Ghent University – imec.*
[2]*TU Wien*

Abstract

With the finalization of RDF 1.2 and SPARQL 1.2, the new triple-term construct brings native reification to RDF and narrows the long-standing gap with Property Graphs. Native support, however, does not imply uniform adoption: existing and new datasets continue to model statement-level metadata through established patterns, so federated queries must mediate between these patterns and the new triple-term syntax. We address this by rewriting queries written against triple terms into queries over the reification pattern actually used by the source, expressing the relationship between patterns and triple terms as a SPARQL CONSTRUCT query that is unfolded into the user's query as a Global-As-View mapping. This paper demonstrates that approach through a web interface in which users author their own mappings, select sources, and observe how their queries are rewritten and executed across federated endpoints. Because both the mappings and the rewriting live entirely within SPARQL, the rewritten query runs on any standards-compliant SPARQL 1.2 federation engine, with no engine modification and with extensibility to reification schemes not foreseen at design time. This makes RDF 1.2 interoperability achievable today using only the query language practitioners already know, and reframes RDF-to-RDF mapping as a declarative, interpretable activity rather than a handwritten algorithm. Looking ahead, we plan to lift current restrictions, formalize and optimize our approach, and turn the language gaps we encounter into concrete input for the upcoming SPARQL maintenance-mode standardization. [3]

Keywords

Data Integration, Query Rewriting, SPARQL 1.2, RDF 1.2, Structural Rewriting, Global-As-View

1. Introduction

With the finalization of the RDF 1.2 [1] and SPARQL 1.2 [2] specifications, data publishers and consumers gain access to the newly introduced *triple term*: a triple that may itself appear in the object position of another triple, allowing arbitrary nesting. This extension brings native reification to RDF and narrows the gap between RDF and Property Graphs [3]. The construct directly descends from RDF-star and SPARQL-star [4], proposed over a decade ago.

Native support, however, does not imply uniform adoption. Existing, and even new, datasets might choose to continue modelling statement-level metadata using established reification patterns: 1. rdf-reification, 2. singleton properties, 3. named graphs, 4. *n*-ary relations, and 5. potentially the vocabulary proposed in the draft RDF 1.2 interoperability note [5]. SPARQL federation [6], which allows querying multiple datasets through a single query, must therefore mediate between these patterns and the new triple-term syntax of SPARQL 1.2.

We address this by rewriting queries written against triple terms into queries over the reification pattern actually used by the source. Our contribution is a generic rewriting approach in which the relationship between each reification scheme and triple terms is described by a standard SPARQL CONSTRUCT query. The CONSTRUCT body is then unfolded into the user's query — no engine modification required. Although we focus on the triple-term-to-legacy direction motivated by the RDF 1.2 transition, the same mechanism applies to the general RDF-to-RDF mapping problem.

The next section reviews related work on mapping and rewriting for RDF. Section 3 describes our naive rewriting approach and it's limitations. Section 4 presents the demonstration, and Section 5 concludes our paper.

2. Related Work

An imperative rewriting algorithm, such as proposed in the RDF 1.2 Interoperability note [5], could handle a fixed set of reification schemes, but a declarative approach offers three advantages: 1. extensibility to reification schemes not foreseen at design time, 2. interpretability of the mappings themselves, and 3. reuse of existing SPARQL querying skills. We therefore step back from a handwritten algorithm and draw on the data integration literature.

[3] Canonical version: https://2026-amw-rewriting.jitsedesmet.be/

AMW 2026: 17th Alberto Mendelzon International Workshop on Foundations of Databases and the Web, November 9th–13th, 2026, Arequipa, Peru
jitse.desmet@ugent.be (J. De Smet); maxime.jakubowski@tuwien.ac.at (M. Jakubowski); ruben.taelman@ugent.be (R. Taelman)
0009-0002-6513-5013 (J. De Smet); 0000-0002-7420-1337 (M. Jakubowski); 0000-0001-5118-256X (R. Taelman)

2.1. Global-As-View And Local-As-View

Data integration is classically framed in terms of mappings between a global schema and local sources[7]. Mappings can either describe the global schema as a view over the local sources (Global-As-View, GAV) or describe each local source as a view over the global schema (Local-As-View, LAV). In both cases, the user writes a query over the global schema while only the source schemata are populated; the system must rewrite the query so that it targets the sources.

RDF complicates this picture in two ways. First, RDF is schemaless: in relational terms, every dataset is a single ternary triples relation. Second, RDF datasets compose naturally through the merge operation [8], a near-union with blank-node renaming. Because the merge is so natural, GAV mappings over RDF are rarely written as a single global union; authors typically write one rule per source and leave the union implicit. This per-source style is easily mistaken for LAV, where describing each source is the definitional act rather than a convenience.

2.2. RML

The RDF mapping language, RML [9] and the standardized Relational to RDF Mapping Language R2RML [10] are declarative languages for describing mappings into RDF. Each rule expresses **one source's** contribution; the global dataset is obtained by taking the dataset union of all rule outputs — the same union-of-views construction described above.

RML views. The RML Logical Views module [11, 12] of the modular RML spec [13] extends this with views defined over a single existing view or source, optionally joined with other views. RML thus remains aligned with knowledge graph *construction*: it does not support structural redefinitions across multiple sources, such as expressing a dataset D constructed from datasets A, B and C such that $D = (A \cup B) \setminus C$.

RML for RDF to RDF. POD-QUERY [14] uses RML to declare RDF-to-RDF mappings: each rule names a single rml:source together with a query over it, and user queries against the global dataset are rewritten via standard GAV unfolding [7]. Because the mappings are expressed in RML, POD-QUERY inherits RML's single-source limitation. The reliance on RML is also a usability cost: RDF consumers typically know SPARQL, not RML. POD-QUERY additionally does not support RDF 1.2 or SPARQL 1.2, which requires special attention in unfolding.

2.3. OBDA

Ontology-Based Data Access (OBDA) [15, 16] places an ontology, typically an RDF vocabulary enriched with description-logic semantics, between the user and the sources. GAV-style mappings link each source (typically relational) to the ontology, which plays the role of the global schema. At query time, a user query against the ontology (or RDF dataset) is first *reformulated* against the ontology itself, compiling *TBox* knowledge into the query, and then *unfolded* against the mappings to produce queries over the sources. Ontop [17] is the reference open-source implementation.

Only the reformulation step uses description-logic reasoning; while unfolding is a syntactic rewrite against the source mappings. OBDA systems often adopt *lightweight* description logics, the DL-Lite family being the canonical choice, to ensure query reformulation suffices.

Description logics can nonetheless be daunting for query practitioners. When full TBox reasoning is not needed, plain RDF-to-RDF GAV is a useful stepping stone: when the sources are RDF, GAV mappings whose source-side query is SPARQL inherit the full expressiveness of SPARQL, including the SERVICE operator for federating across endpoints. Interestingly, recent work has explored using the SPARQL SERVICE operator to access non-RDF resources by transforming them on the fly, typically by (ephemerally) materializing them [18, 19]. To conclude, plain GAV mediation is often the better fit when the goal is a quick structural bridge across sources rather than reasoning over an ontology.

2.4. Positioning

Our approach lets users declare mappings 1. over multiple sources, through SPARQL federation; 2. in the SPARQL query language they already know; and 3. using RDF 1.2 as both source and target. Federation matters here because it enables mappings that go beyond knowledge graph construction, including set-theoretic combinations across sources such as $D = (A \cup B) \setminus C$. Because the mappings are themselves SPARQL CONSTRUCT queries, and since we perform query *rewriting* instead of ephemeral materialization, the approach requires no engine extension and runs on any standard-compliant SPARQL 1.2 federation engine.

3. Query Rewriting

A GAV-style mapping has a body q_S, a query over a source, and a head g, an assertion to be made in the global. We write the mapping as $g \rightsquigarrow q_S$. In SPARQL CONSTRUCT, the template is the head and the WHERE clause is the body. When the template contains multiple triples patterns, we split the mapping into one rule per triple pattern and combine the rules through a UNION in the WHERE clause. Strictly speaking, only one mapping targets the global [7]; when several are provided, they are implicitly merged using dataset merge [8], although for query evaluation we use bag-union rather than the more expensive set-union. Fig. 1 shows two example mappings expressed as CONSTRUCT queries, and Fig. 2 shows their combination into a single rule with one triple pattern in the head.

Algorithm. The rewriting algorithm unfolds each triple pattern in the user query against any mapping whose head matches it. When the triple pattern carries additional constraints, those constraints are pushed into the body of the mapping. The first triple pattern of Fig. 3, for example, fixes the predicate to rdf:reifies and the object to a triple term whose inner predicate is rdf:type; these constants flow into the unfolded body, replacing ?p by rdf:type in Fig. 4. Mappings whose head cannot match the pattern, for instance because the predicates are incompatible, contribute a FILTER(FALSE) branch, which a downstream optimizer can prune.

```
# Create a mapping from rdf reification to triple terms
CONSTRUCT { ?t rdf:reifies <<( ?s ?p ?o )>> }
WHERE {
  ?t a rdf:Statement ; rdf:Subject ?s ;
     rdf:Predicate ?p ; rdf:Object ?o ;
} # and copy all triples
CONSTRUCT { ?s ?p ?o } WHERE {
  ?s ?p ?o .
}
```

Fig. 1: Two mappings: one constructs triple terms in the global, the other passes source triples through unchanged.

```
CONSTRUCT { ?s ?p ?o } WHERE {
  { ?m_t a rdf:Statement ;
      rdf:Subject ?m_s ;
      rdf:Predicate ?m_p ;
      rdf:Object ?m_o .
    BIND(?m_t as ?s) .
    BIND(rdf:reifies as ?p) .
    BIND(triple(?m_s, ?m_p, ?m_o) as ?o). }
  UNION
  { ?s ?p ?o }
}
```

Fig. 2: The same global as Fig. 1, expressed as a single CONSTRUCT in accordance with Subsection 2.1.

```
SELECT * WHERE {
  ?t rdf:reifies <<( ?s rdf:type ?o )>> ;
     ex:confidence ?score .
  FILTER(?score > 0.8)
}
```

Fig. 3: Example user query: triple terms asserting rdf:type with a confidence score above 0.8.

```
SELECT ?score ?o ?s ?t WHERE {
  { ?t rdf:type rdf:Statement ; rdf:Subject ?s ;
      rdf:Predicate rdf:type ; rdf:Object ?o . }
  UNION { # Assume 1.1 sources -> no triple terms
    FILTER ( FALSE ) }
  { # Predicate rdf:reifies does not match ex:confidence
    FILTER ( FALSE ) }
  UNION { ?t ex:confidence ?score }
  FILTER ( ?score > 0.8 )
}
```

Fig. 4: Cleaned-up version (for readability) of the query in Fig. 3 rewritten against the mapping in Fig. 2. The rewritten query no longer asks the source for triple terms; instead it expresses the same intent through the reification scheme used by the source.

Limitations. Blank nodes are not supported in mapping queries. Both the template of a CONSTRUCT query and the BNODE expression introduce fresh blank nodes scoped to a single query solution, which would make the unfolded query unstable. We plan to support *Skolemized blank nodes* in future work.

Blank nodes in the sources are likewise unsupported. As in standard SPARQL federation when the same endpoint is queried more than once, blank nodes from a source cannot be matched across invocations, since blank nodes are scoped to the document or result set in which they are serialized [2]. Skolemization of the source is therefore a prerequisite.

User queries that use the recursive property path operators * and + are not supported. We expect that a recursive extension to SPARQL will be a prerequisite for handling them.

4. Demonstration

The demo lets users issue federated queries over sources of their choice, against a global schema that they define through their own mappings. The interface consists of four panels.

The *mappings panel* (top left) is where users manage the CONSTRUCT queries that together define the global schema. Formally, a GAV mapping is a single CONSTRUCT (see Subsection 2.1); the UI lets users write multiple mappings whose outputs are implicitly unioned.

The *query panel* (top right) is where users select sources and write the query. With no sources selected, the rewritten query can still return results when the mappings include SERVICE calls. When many sources are selected, the federation engine performs exhaustive source selection [20], which can itself be viewed as a second GAV rewriting in which every mapping has the form CONSTRUCT { ?s ?p ?o } WHERE { SERVICE <serviceA> { ?s ?p ?o } }. The panel also offers preset scenarios: 1. trustworthiness using RDF 1.2, 2. exhaustive source selection, and 3. spam-filter.

The *rewritten query panel* (bottom left) is read-only; after execution it shows the user query unfolded against the mappings.

The *results panel* (bottom right) reports elapsed execution time, the result count, and the streaming bindings as they arrive.

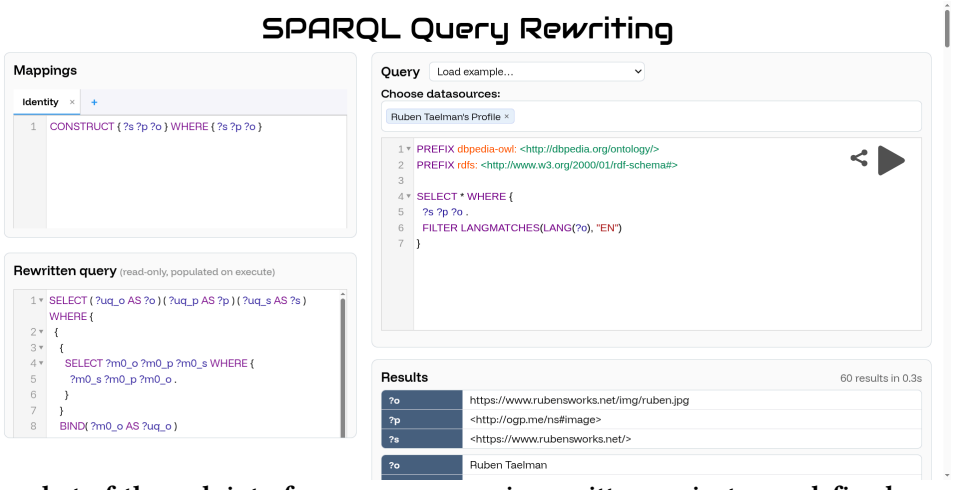

Fig. 5: Screenshot of the web interface: a user query is rewritten against user-defined mappings and executed over the chosen sources.

5. Conclusion

We have presented a preliminary implementation of a generic framework for RDF 1.2 interoperability through query rewriting, and, more broadly, for declarative RDF-to-RDF mapping. The approach is declarative rather than imperative, a choice that buys 1. extensibility to reification schemes not foreseen at design time, 2. interpretable transformations, and 3. reuse of existing SPARQL skills.

The mappings are GAV rules expressed as SPARQL CONSTRUCT queries, and rewriting proceeds by unfolding those rules into the user's query. Because both the rules and the rewriting live within SPARQL, the rewritten query runs on any standards-compliant SPARQL 1.2 federation engine. When a desired mapping cannot be expressed in standard SPARQL, the missing feature becomes concrete input for the upcoming maintenance-mode standardization of the language. We have already encountered a handful of candidates: a CONSTRUCT GRAPH form, syntactic sugar for triple-term templates, and recursive view definitions.

This is ongoing work; performance optimization and concrete language-extension proposals are the immediate next steps. Our aim with this demonstration is to draw attention to GAV-style SPARQL mappings, to showcase their

use for RDF 1.2 interoperability, and to open discussion on the language features the community needs.

Acknowledgements

Jitse De Smet is a predoctoral fellow of the Research Foundation – Flanders (FWO) (1SB8525N).

Declaration On Generative AI

During the preparation of this work, the author(s) used Claude Opus 4.7 in order to: Paraphrase and reword, improve writing style, perform grammar and spelling checks, and critically review the text written by the authors to highlight essential information. After using these tool(s)/service(s), the author(s) reviewed and edited the content as needed and take(s)full responsibility for the publication's content.

References

[1] Kellogg, G., Seaborne, A., Champin, P.-A., Hartig, O.: RDF 1.2 Concepts and Abstract Syntax. W3C, https://www.w3.org/TR/rdf12-concepts/ (2025).
[2] Hartig, O., Seaborne, A., Taelman, R., Williams, G., Tanon, T.P.: SPARQL 1.2 Query Language. https://www.w3.org/TR/sparql12-query/ (2025).
[3] Information technology — Database languages — GQL. International Organization for Standardization, Geneva, CH, https://www.iso.org/standard/76120.html (2024).
[4] Hartig, O.: Foundations of RDF* and SPARQL*:(An alternative approach to statement-level metadata in RDF). In: AMW 2017 11th Alberto Mendelzon International Workshop on Foundations of Data Management and the Web, Montevideo, Uruguay, June 7-9, 2017. Juan Reutter, Divesh Srivastava (2017).
[5] Champin, P.-A.: RDF 1.2 Interoperability. W3C, https://www.w3.org/TR/rdf12-interop/ (2026).
[6] Williams, G., Taelman, R.: SPARQL 1.2 Federated Query. W3C, https://www.w3.org/TR/sparql12-federated-query/ (2024).
[7] Lenzerini, M.: Data integration: A theoretical perspective. In: Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems. pp. 233–246 (2002).
[8] Haudebourg, T., Arndt, D., Patel-Schneider, P.: RDF 1.2 Semantics. W3C, https://www.w3.org/TR/rdf12-semantics/ (2025).
[9] Dimou, A., Vander Sande, M., Colpaert, P., Verborgh, R., Mannens, E., Van de Walle, R.: RML: A generic language for integrated RDF mappings of heterogeneous data. Ldow. 1184, (2014).
[10] Sundara, S., Das, S., Cyganiak, R.: R2RML: RDB to RDF Mapping Language. W3C, https://www.w3.org/TR/r2rml/ (2012).
[11] Maria, P., de Vleeschauwer, E., Lanti, D.: RML Logical Views. W3C, https://kg-construct.github.io/rml-lv/spec/docs/ (2026).
[12] de Vleeschauwer, E., Maria, P., De Meester, B., Colpaert, P.: RML-view-to-CSV : a proof-of-concept Implementation for RML Logical Views. In: KGCW 2024 : Knowledge Graph Construction 2024 : Proceedings of the 5th International Workshop on Knowledge Graph Construction co-located with 21th Extended Semantic Web Conference (ESWC 2024). p. 14 (2024).
[13] Maria, P., Dimou, A.: RML-Core. W3C, https://kg-construct.github.io/rml-core/spec/docs/ (2026).
[14] Vandenbrande, M., JAKUBOWSKI, M., Bonte, P., Buelens, B., Ongenae, F., BUSSCHE, J.V.A.N.D.E.N.: POD-QUERY: Schema Mapping and Query Rewriting for Solid Pods. In: Proceedings of the Second International Workshop on Semantic Industrial Information Modelling (SemIIM 2023) (2023).
[15] Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. In: Journal on data semantics X. pp. 133–173. Springer (2008).
[16] Xiao, G., Calvanese, D., Kontchakov, R., Lembo, D., Poggi, A., Rosati, R., Zakharyaschev, M.: Ontology-based data access: A survey. Presented at the (2018).
[17] Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., Xiao, G.: Ontop: Answering SPARQL queries over relational databases. Semantic Web. 8, 471–487 (2016).
[18] Asprino, L., Daga, E., Gangemi, A., Mulholland, P.: Knowledge graph construction with a façade: A unified method to access heterogeneous data sources on the web. ACM Transactions on Internet Technology. 23, 1–31 (2023). doi:10.1145/3555312
[19] Hartig, O., Westman, J.: Querying Federations of SPARQL Endpoints and JSON-based Web APIs with HeFQUIN. In: Proceedings of Satellite Events of the 23rd European Semantic Web Conference (ESWC), Lecture Notes in Computer Science, Springer (2026).
[20] Cheng, S., Hartig, O.: FedQPL: A Language for Logical Query Plans over Heterogeneous Federations of RDF Data Sources. In: Indrawan-Santiago, M., Pardede, E., Salvadori, I.L., Steinbauer, M., Khalil, I., and Kotsis, G. (eds.) iiWAS '20: The 22nd International Conference on Information Integration and Web-based Applications & Services, Virtual Event / Chiang Mai, Thailand, November 30 - December 2, 2020. pp. 436–445. ACM (2020). doi:10.1145/3428757.3429120