

Demonstration of RDF and SPARQL 1.2 Interoperability through Query Rewriting

Jitse De Smet¹, Maxime Jakubowski²

¹IDLab, Department of Electronics and Information Systems, Ghent University – imec, Belgium

²TU Wien, Austria

Abstract

With the finalization of RDF 1.2 and SPARQL 1.2, the new triple-term construct brings native reification to RDF. Native support, however, does not imply uniform adoption: existing datasets continue to model statement-level metadata through established reification patterns, so federated queries must mediate between these patterns and the new syntax. We address this by rewriting queries written against triple terms into queries over the reification pattern actually used by each source, expressing the relationship as a SPARQL CONSTRUCT query that is unfolded into the user's query. This paper demonstrates the 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. Reification translation is one application of a more general framework for SPARQL query answering under declarative RDF-to-RDF mappings, which we plan to formalize and optimize. The language gaps it exposes also give us concrete input for the upcoming SPARQL maintenance-mode standardization.^{1 2 3}

Keywords

SPARQL 1.2, RDF 1.2, Reification, Metadata, Data Integration, Query Rewriting

1. Introduction

With the finalization of the recent RDF 1.2 [1] and SPARQL 1.2 [2] specifications, data publishers and consumers will gain access to a newly introduced extension of the RDF data model: the *triple term*. This extension explicitly addresses the need to have a uniform way of modelling statement-level metadata, which previously was done with *reification patterns*. Triple terms enable arbitrary nesting of triples, bringing native reification to RDF and narrowing the gap between RDF and Property Graphs [3]. The construct directly descends from RDF-star and SPARQL-star [4], proposed over a decade ago.

However, standardization does not imply uniform adoption. Existing, and even new, datasets might choose to continue modelling statement-level metadata using established reification patterns (e.g., rdf-reification [5], *n*-ary relations, or potentially the vocabulary proposed in the draft RDF 1.2 interoperability note [6]). As a result, querying RDF means handling all of these patterns, even though triple terms now provide a single standard. The problem worsens in federated settings, where sources may use different patterns, and the query engine must mediate between them.

We address this gap with two contributions. First, we propose query rewriting as a practical route to interoperability: SPARQL 1.2 queries written against triple terms are rewritten into queries over the reification pattern actually used by each source, with per-source mappings handling federated heterogeneity. Second, we express these mappings declaratively as standard SPARQL CONSTRUCT queries, keeping the entire solution within the RDF ecosystem and requiring no new mapping language. The rewriting algorithm itself builds on classic unfolding techniques, applied to RDF 1.2.

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

¹Video: <https://cloud.ilabt.imec.be/index.php/s/mm5ZipXsSkMDw5f>

²Canonical version: <https://2026-semantic-rewriting.jitsedesmet.be/>

³Demonstration: <https://2026-query-rewriting.demo.jitsedesmet.be/>

Posters, Demos, Blue Sky, and Tutorials at SEMANTiCS 2026, Sep 2026, Ghent, Belgium

✉ jitse.desmet@ugent.be (J. D. Smet); maxime.jakubowski@tuwien.ac.at (M. Jakubowski)

🌐 <https://jitsedesmet.be/> (J. D. Smet); <https://www.mjakubowski.info/> (M. Jakubowski)

🆔 0009-0002-6513-5013 (J. D. Smet); 0000-0002-7420-1337 (M. Jakubowski)



© 2026 Copyright for this paper by its authors. Use permitted under Creative Commons License Attribution 4.0 International (CC BY 4.0).

2. Related work

An imperative rewriting algorithm, such as the one proposed in the RDF 1.2 Interoperability note [6], could handle a fixed set of reification schemes. A declarative approach offers three advantages over such an algorithm: 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 an imperative algorithm and draw on the data integration literature, where a Global-As-View (GAV) mapping describes a global schema as a view over the local sources, and user queries against the global are unfolded into queries over the sources [7].

The closest prior work is POD-QUERY [8], which applies GAV unfolding to RDF-to-RDF mappings. Each mapping rule names a single `rml:source` paired with a query over it, expressed in RML [9]. Because RML rules are scoped to one source, POD-QUERY inherits a single-source limitation: set-theoretic combinations across sources A , B and C , such as a global $D = (A \cup B) \setminus C$, cannot be expressed as a single mapping. We instead write mappings as SPARQL CONSTRUCT queries, which can federate across sources through the SERVICE operator and so express such combinations directly. Expressing mappings in SPARQL rather than RML also lowers the barrier for RDF consumers, who typically know the former but not the latter. POD-QUERY additionally predates RDF 1.2 and SPARQL 1.2, and so does not address the triple-term rewriting that motivates this work.

A related line of work is Ontology-Based Data Access (OBDA) [10, 11], whose reference implementation is Ontop [12]. OBDA also relies on GAV unfolding, but typically maps relational sources – through mapping languages such as R2RML [13] – to an ontology and layers description-logic reasoning on top. When the sources are already RDF and full TBox reasoning is not needed, plain GAV mediation is a lighter-weight fit, and the SPARQL SERVICE operator provides the federation across sources for free.

3. Query rewriting

Figure 1 gives an overview of our approach: a user query is sent to a federation mediator, which rewrites it through one or more user-authored mappings into queries against the sources. We express each mapping as a SPARQL CONSTRUCT query: the template (mapping head) states what the mapping asserts about the global, and the WHERE clause (mapping body) queries the source. When several mappings target the same global, their outputs are implicitly unioned, in line with RDF’s dataset merge operation [14]. Figure 2 shows two such mappings: one constructs triple terms from `rdf:reifies` statements in the source, the other passes source triples through unchanged. When a CONSTRUCT template contains multiple triple patterns, we split it into one rule per triple pattern and combine them through a UNION in the WHERE clause as required by GAV-mapping theory; Figure 3 shows the equivalent single-headed form.

Algorithm. The rewriting algorithm unfolds each triple pattern in the user query against any mapping whose template (mapping head) matches it, pushing additional constraints from the pattern into the WHERE clause of the matching mapping. Consider the user query in Figure 4, which asks for triple terms whose inner predicate is `rdf:type` together with a confidence score above 0.8. The first triple pattern fixes the predicate to `rdf:reifies` and the inner predicate of the triple term to `rdf:type`; these constants flow into the unfolded body, replacing `?p` by `rdf:type` in Figure 5. Mappings whose template cannot match a pattern – for instance because the predicates are incompatible – contribute a `FILTER(FALSE)` branch, which a downstream optimizer can prune.

Scope. Although this paper focuses on RDF 1.2 interoperability, the rewriting framework is more general: we envision it as a broader approach to SPARQL query answering under mappings, of which reification-pattern translation is one application. Because the rewritten query is itself a federated SPARQL query, the approach inherits the boundaries of federated SPARQL. Blank nodes have limited support on both sides of a mapping: CONSTRUCT templates and the `BNODE` expression introduce fresh blank nodes scoped to a single query solution, and blank nodes returned by a source cannot be matched across invocations of the same endpoint, since blank nodes are scoped to the document or result set in

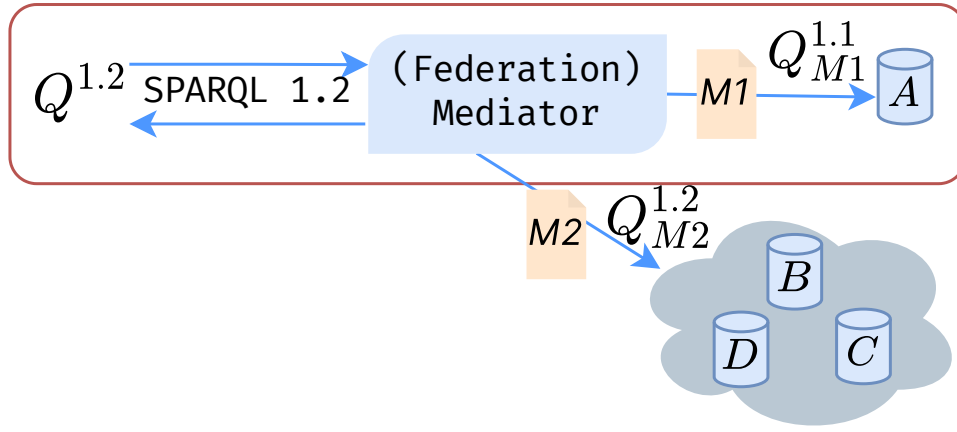


Figure 1: A SPARQL 1.2 query $Q^{1.2}$ enters the Federation Mediator, which rewrites it through user-authored mappings. The red box highlights the reification interoperability use-case: mapping $M1$ rewrites the query into $Q_{M1}^{1.1}$ against a single legacy source A that uses an established reification pattern. The full figure depicts the general setting: $M2$ is a GAV mapping spanning multiple sources B, C, D , producing a federated query $Q_{M2}^{1.2}$. Each mapping is itself a SPARQL CONSTRUCT query; the mediator unfolds them into $Q^{1.2}$ and the rewritten query runs on any standards-compliant SPARQL 1.2 federation engine.

```
# 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 .
}
```

Figure 2: 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 }
}
```

Figure 3: The same global as Figure 2, expressed as a single CONSTRUCT required for the GAV-unfolding algorithm.

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

Figure 4: 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 )
}
```

Figure 5: Cleaned-up version (for readability) of the query in Figure 4 rewritten against the mapping in Figure 3. 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.

which they are serialized [2]. Skolemization is the classical solution. Recursive property paths (`*` and `+`) are likewise not handled, pending a recursive extension to SPARQL.

SPARQL Query Rewriting

The screenshot displays the SPARQL Query Rewriting web interface, which is divided into four main panels:

- Mappings:** A panel on the top left showing a list of mappings. The first mapping is selected, showing a triple pattern: `PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>`, `CONSTRUCT {`, `?t rdf:reifies << ?s ?p ?o >>`, `} WHERE {`, `?t a rdf:Statement; rdf:subject ?s;`, `rdf:predicate ?p; rdf:object ?o .`, `}`.
- Query:** A panel on the top right where a user query is entered. The query is: `PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>`, `PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>`, `PREFIX up: <http://purl.uniprot.org/core/>`, `PREFIX uniprotkb: <http://purl.uniprot.org/uniprot/>`, `SELECT * WHERE {`, `<< uniprotkb:P01308 up:annotation ?annotation >> up:attribution ?attribution .`, `# ?attribution up:evidence ?evidence ; # ECO code (e.g. ECO_0000269 = experimental)`, `# up:source ?source . # citation / PubMed entry`, `}`, `LIMIT 1`.
- Rewritten query:** A panel on the bottom left showing the rewritten query: `SELECT ?uq_annotation ?uq_attribution WHERE {`, `{`, `SELECT ?m0_o ?m0_t WHERE {`, `?m0_o <http://www.w3.org/1999/02/22-rdf-syntax-ns#type> <http://www.w3.org/1999/02/22-rdf-syntax-ns#Statement> .`, `?m0_t <http://www.w3.org/1999/02/22-rdf-syntax-ns#subject> <http://purl.uniprot.org/uniprot/P01308> .`, `?m0_t <http://www.w3.org/1999/02/22-rdf-syntax-ns#predicate> <http://purl.uniprot.org/core/annotation> .`, `}`, `}`.
- Results:** A panel on the bottom right showing the results of the query. It displays two rows of results: `?uq_annotation` with value `<http://purl.uniprot.org/uniprot/P01308#SIP0A098500B0F5E9FA>` and `?uq_attribution` with value `<http://purl.uniprot.org/uniprot/P01308#attribution-B6DD287E43423D231ED8572818879420>`. The results are shown in a table format.

Figure 6: Screenshot of the web interface: a user query is rewritten against user-defined mappings and executed over the chosen sources.

4. Demonstration

The demo’s main scenario is RDF 1.2 interoperability: the user writes a SPARQL 1.2 query using triple terms, points it at one or more sources that model statement-level metadata through legacy reification patterns, and sees the query rewritten so that it runs against the sources as-is. The rewritten query is shown next to the original, so users can inspect how it was transformed using Traquila [15], and then executed using Comunica [16]. Alongside this main scenario, the demo ships with several preset scenarios that exercise different facets of the framework: 1. wikidata reification, 2. exhaustive source selection, 3. uniprot reification, and 4. paper filter Verborgh.

In the Wikidata reification example, we construct the global by copying all existing triples and adding triple terms to statements following Wikidata’s n -ary reification approach; we then query the *instance of* property of Belgium. The exhaustive source selection example runs over the RDF profiles of Bryan, Ruben, and Jitse, querying the names of all scholarly articles on their profiles. The UniProt example again copies all existing triples and adds triple terms, this time following the standard RDF reification that UniProt uses; we then query the annotations of a given protein. Finally, the set-theoretic example shows how these combinations work in practice: we construct the global as every triple in Jitse’s and Ruben’s profiles, except those related to articles that Verborgh also describes. Querying the scholarly articles and their names in this global, we see that Jitse’s vision paper is missing, for instance, because Verborgh co-authored it.

The query rewriting mechanism is not limited to reification translation. Any mapping the user writes becomes part of the global schema, and the rewriting handles it identically. Federation across sources is itself a special case: selecting a set of endpoints is equivalent to a second layer of GAV rewriting in which every mapping has the form `CONSTRUCT { ?s ?p ?o } WHERE { SERVICE <s> { ?s ?p ?o } }`, which the engine performs as exhaustive source selection [17]. The demo thus stands in for a broader claim from Section 3: that mapping-driven query rewriting is a general way to query RDF under user-defined schemas, with reification interoperability as one application.

The interface exposes this through four panels. The *mappings panel* (top left) is where the user writes CONSTRUCT queries that together define the global schema. The *query panel* (top right) is where the user selects sources and writes the user query against that schema. The *rewritten query panel* (bottom

left) is read-only and shows the user query unfolded against the mappings after execution. The *results panel* (bottom right) reports elapsed execution time, the result count, and the streaming bindings as they arrive.

5. Conclusion

We have presented a preliminary implementation of query rewriting for RDF 1.2 interoperability, addressing the gap between the new triple-term construct and the reification patterns that datasets continue to use in practice. This use-case is one instantiation of a more general vision: SPARQL query answering over declarative RDF-to-RDF mappings, of which reification translation is one application among many.

The mappings are GAV rules expressed as SPARQL CONSTRUCT queries, and rewriting proceeds by unfolding those rules into the user’s query. This declarative approach buys three things over a handwritten algorithm: 1. extensibility to reification schemes not foreseen at design time; 2. interpretable transformations that users can read directly; and 3. reuse of existing SPARQL skills, since mappings are written in the same language as the queries they rewrite. Because both the rules and the rewriting live within SPARQL, the rewritten query runs on any standards-compliant SPARQL 1.2 federation engine, with no engine modification required.

This is ongoing work. Immediate next steps include performance optimization, and broader applications beyond reification translation. A separate thread is concrete input to the upcoming maintenance-mode standardization of SPARQL: when a desired mapping cannot be expressed in standard SPARQL, the missing feature becomes a candidate language extension. We have already encountered a handful: a CONSTRUCT GRAPH form, syntactic sugar for triple-term templates, and recursive view definitions. 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 to make the broader vision practical.

Acknowledgments

This publication is based upon work from COST Action CA23147 GOBLIN—Global Network on Large-Scale, Cross-domain and Multilingual Open Knowledge Graphs, supported by COST (European Cooperation in Science and Technology). 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] G. Kellogg, A. Seaborne, P.-A. Champin, O. Hartig, RDF 1.2 Concepts and Abstract Syntax, W3C Working Draft, W3C, 2025. URL: <https://www.w3.org/TR/rdf12-concepts/>, <https://www.w3.org/TR/2025/WD-rdf12-concepts-20250225/>.
- [2] O. Hartig, A. Seaborne, R. Taelman, G. Williams, T. P. Tanon, SPARQL 1.2 Query Language, W3C Working Draft, W3C, 2025. URL: <https://www.w3.org/TR/sparql12-query/>.

- [3] ISO/IEC 39075:2024, Information technology – Database languages – GQL, Standard, International Organization for Standardization, Geneva, CH, 2024. URL: <https://www.iso.org/standard/76120.html>.
- [4] O. Hartig, Foundations of RDF $\star$ and SPARQL $\star$ (An Alternative Approach to Statement-Level Metadata in RDF), in: AMW, CEUR Workshop Proceedings, CEUR-WS.org, 2017.
- [5] R. Guha, D. Brickley, RDF Schema 1.1, W3C Recommendation, W3C, 2014. URL: <https://www.w3.org/TR/rdf11-schema/>, <https://www.w3.org/TR/2014/REC-rdf-schema-20140225/>.
- [6] P.-A. Champin, RDF 1.2 Interoperability, W3C Note Draft, W3C, 2026. URL: <https://www.w3.org/TR/rdf12-interop/>.
- [7] M. Lenzerini, Data integration: A theoretical perspective, in: PODS, ACM, 2002, pp. 233–246.
- [8] M. Vandenbrande, M. Jakubowski, P. Bonte, B. Buelens, F. Ongenae, J. Van den Bussche, POD-QUERY: schema mapping and query rewriting for Solid Pods, in: ISWC (Posters/Demos/Industry), CEUR Workshop Proceedings, CEUR-WS.org, 2023.
- [9] A. Dimou, M. V. Sande, P. Colpaert, R. Verborgh, E. Mannens, R. V. de Walle, RML: A generic language for integrated RDF mappings of heterogeneous data, in: LDOW, CEUR Workshop Proceedings, CEUR-WS.org, 2014.
- [10] A. Poggi, D. Lembo, D. Calvanese, G. D. Giacomo, M. Lenzerini, R. Rosati, Linking data to ontologies, *J. Data Semant.* 10 (2008) 133–173.
- [11] G. Xiao, D. Calvanese, R. Kontchakov, D. Lembo, A. Poggi, R. Rosati, M. Zakharyashev, Ontology-based data access: A survey, in: IJCAI, *ijcai.org*, 2018, pp. 5511–5519.
- [12] D. Calvanese, B. Cogrel, S. Komla-Ebri, R. Kontchakov, D. Lanti, M. Rezk, M. Rodriguez-Muro, G. Xiao, Ontop: Answering SPARQL queries over relational databases, *Semantic Web* 8 (2017) 471–487.
- [13] S. Sundara, S. Das, R. Cyganiak, R2RML: RDB to RDF Mapping Language, W3C Recommendation, W3C, 2012. URL: <https://www.w3.org/TR/r2rml/>, <https://www.w3.org/TR/2012/REC-r2rml-20120927/>.
- [14] P. Patel-Schneider, D. Arndt, E. Franconi, RDF 1.2 Semantics, W3C Working Draft, W3C, 2025. URL: <https://www.w3.org/TR/rdf12-semantics/>.
- [15] J. De Smet, R. Taelman, Traquila: Providing a foundation for the evolving SPARQL ecosystem through modular query parsing, transformation, and generation, in: The Semantic Web - 23rd European Semantic Web Conference, ESWC 2026, 2026, pp. 232–252. doi:10.1007/978-3-032-25159-6_13.
- [16] R. Taelman, J. V. Herwegen, M. V. Sande, R. Verborgh, Comunica: A modular SPARQL query engine for the web, in: D. Vrandečić, K. Bontcheva, M. C. Suárez-Figueroa, V. Presutti, I. Celino, M. Sabou, L. Kaffee, E. Simperl (Eds.), The Semantic Web - ISWC 2018 - 17th International Semantic Web Conference, Monterey, CA, USA, October 8–12, 2018, Proceedings, Part II, volume 11137 of *Lecture Notes in Computer Science*, Springer, 2018, pp. 239–255. doi:10.1007/978-3-030-00668-6_15.
- [17] S. Cheng, O. Hartig, FedQPL: A language for logical query plans over heterogeneous federations of RDF data sources, in: iiWAS, ACM, 2020, pp. 436–445. URL: <https://liu.diva-portal.org/smash/get/diva2:1594213/FULLTEXT01.pdf>.