Towards Hybrid Link Traversal: Challenges and Research Directions for Heterogeneous Dataspaces

Ruben Eschauzier[1], Ruben Taelman[1]

[1]Department of Electronics and Information Systems, Ghent University – imec

Abstract

Decentralized dataspaces preserve data sovereignty by keeping data at its source, but querying a large number of autonomous sources with heterogeneous interfaces introduces significant challenges. Traditional federated engines fail to scale to this size, while existing Link Traversal-based Query Processing (LTQP) systems can only handle Linked Data documents. By ignoring the (query) capabilities of expressive interfaces, such as SPARQL endpoints, current LTQP engines suffer performance issues. To address this, we advocate for Hybrid LTQP, an execution strategy that combines the dynamic runtime discovery of link traversal with the performance benefits of delegating complex sub-queries to capable server-side interfaces. This paper reviews the state-of-the-art in hybrid traversal and identifies the critical challenges hindering its implementation: establishing exclusive groups without prior knowledge, integrating dynamic sub-queries into active query plans, enabling reliable service discovery, and deduplicating alternative architectural views. Solving the challenges identified in this paper is a strict prerequisite for making hybrid decentralized querying practically viable across heterogeneous dataspaces. Consequently, future research must prioritize these open problems to engineer the next generation of scalable, interface-aware LTQP query engines.

Keywords

Link Traversal-based Query Processing, Hybrid Link Traversal, SPARQL, Dataspaces, Query Optimization

This paper addresses (Issue #2) of the W3C Dataspaces Community Group.

1. Introduction

While centralizing data into warehouses or lakes can benefit query engine performance, it is antithetical to the core principles of *dataspaces*. Dataspaces promote a decentralized ecosystem where data remains at the source, managed by independent participants under agreed-upon governance models. This architectural shift introduces significant technical challenges for query engines, which must now discover and query data across a vast network of sources, each enforcing its own usage policies.

Centralized querying requires collecting large volumes of proprietary or sensitive data in a single source, which conflicts with the requirements for fine-grained sovereignty and minimized data replication. Conversely, traditional *federated* approaches [1, 2, 3] support decentralization but typically assume a static federation of a small number (10–100) of uniform, expressive endpoints (e.g., SPARQL endpoints) [4]. Closely related are heterogeneous federation approaches [5, 6, 3], which operate similarly but allow different source interfaces, such as Triple Pattern Fragments (TPF) [7], Bindings-Restricted TPF (brTPF) [8], Star Pattern Fragments (SPF) [9], SaGe [10], smart-KG [11], and WiseKG [12]. Enforcing complex usage policies [13] on such heavyweight interfaces significantly impacts performance [14], and these engines struggle to scale to the thousands of sources characteristic of a dataspace.

In contrast, realistic dataspace environments are characterized by a massive scale of permissioned sources and high interface diversity [15, 16]. Beyond the structured fragments managed by traditional heterogeneous federated engines, dataspace participants may, for example, expose data through generic HTTP documents or derived views [17]. Because these participants are autonomous and decentralized, their availability and specific query capabilities are often unknown until the point of execution. Addressing this requires a *hybrid* federation approach: one that can dynamically discover and interoperate

The Third International Workshop on Semantics in Dataspaces, co-located with the Extended Semantic Web Conference, June 01, 2025, Portorož, Slovenia
ruben.eschauzier@ugent.be (R. Eschauzier); ruben.taelman@ugent.be (R. Taelman)
0000-0002-6475-806X (R. Eschauzier); 0000-0001-5118-256X (R. Taelman)

with highly heterogeneous sources at runtime, without the overhead of prior centralization or static configuration.

Link Traversal-based Query Processing (LTQP) is an approach that meets these needs by discovering data sources during execution via hypermedia links found in earlier results. Starting from seed references (e.g., a self-description or catalog entry), it follows links asynchronously. While LTQP is traditionally defined as an approach for querying over federated Linked Data documents by following links at runtime, data within a dataspace is often exposed through diverse access interfaces that may not support standard document-based traversal and querying.

We advocate for *Hybrid* LTQP, which we define below

Definition (Hybrid LTQP): We define *Hybrid Link Traversal Query Processing* as a query execution paradigm that advances beyond traditional Linked Data Document-centric traversal. It enables query engines to traverse, integrate, and actively leverage the computational capabilities of heterogeneous, expressive data interfaces alongside standard Linked Data Documents.

Examples of such expressive interfaces are SPARQL endpoints, materialized views, and TPF [7].

To leverage these varied levels of expressivity, a hybrid engine must offload computation to the server whenever possible. For instance, a SPARQL endpoint can execute complex sub-queries locally, while document-oriented interfaces can't. By leveraging internal indexing and local data knowledge, the endpoint could produce results far more efficiently than a client-side LTQP engine. Offloading these tasks to capable endpoints could significantly reduce overall query latency. However, building such a system introduces several unresolved challenges. In this paper, we review the state-of-the-art in hybrid querying, analyze related work, and outline open issues for future research.

2. Related Work

While Hybrid LTQP has previously been explored in literature, its exact definition varies across studies. Early works [18, 19, 20, 21] typically define a "hybrid" approach as the combination of live link traversal with traditional centralized or federated querying mechanisms. In contrast, our work employs a strictly live-traversal process, redefining the "hybrid" aspect as the traversal over heterogeneous data sources.

2.1. Hybrid Traversal - Local vs Remote

A branch of hybrid link traversal querying uses precomputed indexes, either located within the decentralized environment or computed locally, to speed up query execution or source discovery. The first of such approaches uses precomputed indexes to rerank sources based on relevance to the query's triple patterns and joins, retrieving them to execute queries [22]. Other works [23] propose caching information on sources to help prioritize them.

Another approach is to store entire RDF graphs in local stores to quickly execute parts of the query. As a result, non-blocking index-based operators [24] can be used to leverage the locally computed indexes for faster query execution.

A major limitation is the freshness of the data in the store and indexes, which can quickly go stale for high-velocity data sources. In addition, these preliminary works deviate from our definition of hybrid LTQP, as they rely solely on Linked Data Document-centric traversal to populate local caches or use decentralized indexes only to optimize the Linked Data Document-based traversal process.

2.2. Hybrid Traversal - Multiple Processes

The methods in the literature falling within our definition of hybrid traversal use two separate query processes to obtain data from heterogeneous sources. First, coherence-aware query processing uses a centralized store to obtain initial query results quickly. Using a method called coherence estimation, it determines which data from this store is likely stale. The stale data is then discovered and queried using the traversal-based second process during query execution. Finally, the results are combined to provide the query answer.

A second approach is a hybrid method that executes a federated query over known SPARQL endpoints concurrently with traversal-based query processing over URIs [20]. While this mitigates the issue of data freshness by querying sources at runtime rather than relying on static indexes, it requires the relevant SPARQL endpoints to be configured prior to query execution. Consequently, the system cannot utilize new SPARQL endpoints discovered during the traversal process.

2.3. Hybrid Traversal - Closed Systems

Other approaches integrate into the dataspace itself. ESPRESSO [25] is such an approach, it is a framework around the Solid dataspace specifically that defines several apps in the dataspace to perform certain (optimization) tasks. In the Solid environment, data is stored into personal data vaults. These vaults use the Linked Data Platform [26] to arrange the data into a folder-like structure. To avoid traversing the entire pod to query it, ESPRESSO uses the *Brewmaster* application to locally create indexes of the pod and uses the *CoffeeFilter* application to search its contents. In addition, ESPRESSO uses an overlay network to distribute end-user queries to relevant data resources across Solid servers, where each Solid server is mapped to a federated database node in the overlay network.

Unlike approaches based on link traversal or service discovery within Solid, ESPRESSO relies on an explicitly configured overlay network, with no mechanism for dynamically discovering participating servers or query endpoints. Furthermore, currently ESPRESSO only supports distributed keyword-based search, it does not support SPARQL queries over the sources.

While these approaches do hybrid link traversal, they significantly deviate from our interpretation of hybrid traversal. This is in contrast to the setting described in this paper where the discovered sources themselves are heterogeneous, serendipitously discovered, and cannot be assumed to be present for every source within the dataspace.

2.4. Hybrid Link Traversal over Heterogeneous Sources

State-of-the-art LTQP engines, most notably Comunica Link Traversal [27, 28], demonstrate hybrid capabilities by dynamically federating over discovered SPARQL endpoints, TPF interfaces, and hypermedia documents. However, these systems are currently limited by a triple pattern level-only strategy: they fail to leverage the full expressivity of sophisticated interfaces. For instance, upon discovering a SPARQL endpoint, existing engines restrict interaction to simple *triple pattern* lookups and do not attempt sending sub-queries.

This often results in the client retrieving high volumes of intermediate results for performing local joins, reducing the optimization and latency benefits of more expressive server-side execution.

3. Challenges with Hybrid Link Traversal

In this section, we will outline specific identified problems with hybrid link traversal making implementation difficult.

3.1. Establishing Exclusive Groups

In classical federated query processing, *exclusive groups* are sets of triple patterns within a query that can *only* be answered by a single source [1]. instead of retrieving results for individual triple patterns and joining them locally, the engine can delegate the entire group as a single sub-query (e.g., a Basic Graph Pattern) to the remote endpoint. This technique significantly reduces the number of remote requests and processed intermediate results [29, 1].

The identification of exclusive groups is a strict requirement for this sub-query delegation. If a query engine cannot verify that a group of patterns belongs *exclusively* to a single source, it cannot safely dispatch them as a conjunctive sub-query (BGP). Doing so risks *incomplete results* because the endpoint will execute a server-side join. If the endpoint lacks data for one of the patterns, or if valid join partners reside on a different server, the server-side join execution will filter out intermediary results that could have been successfully joined with data from another source. Thus, to guarantee complete results, a query engine must only delegate conjunctive sub-queries when it can guarantee this sub-query forms an exclusive group.

Federation engines like *FedX* [1], *HiBISCuS* [30], Comunica [27], and *FedUp* [29] rely on either prior knowledge on the content of the federation members, or issue queries (such as ASK) to determine these groups.

For Hybrid Link Traversal to use SPARQL endpoints to compute joins locally, the engine should also identify exclusive groups. The problem is the unbounded nature of LTQP [31]. As during LTQP the query engine does not have access to all sources it will query over, exclusive groups can never be established using traditional federated query approaches. Even if we move to a finite web [31], the engine would have to first traverse all reachable [31] sources before exclusive groups can be determined. This is antithetical to the streaming nature of LTQP engines [27, 32] and will delay time to first result. Thus, traditional approaches for determining exclusive groups are not applicable.

3.1.1. Research Avenues

RT: I'd repeat the main problem here first: what information could we use to safely determine exclusive groups?

Discoverable Indexes: Indexes can be effectively used to catalogue the data sources present in a dataspace. These indexes could be deployed as services, similar to approaches like ESPRESSO [25], pointing towards available source URIs and potentially exposing summary statistics. Using such indexes to quickly determine relevant sources, LTQP engines can apply traditional federation techniques to establish exclusive groups and perform source selection. This would move the problem from traversal to federation, however new challenges such as index completeness and maintenance would arise

Caching: Similar to discoverable indexes, caches provide prior knowledge about the data sources present in a dataspace. Provided the cache maintains a degree of freshness and completeness, the query engine can perform a 'simulated traversal' offline over the cached data to identify relevant sources. Any gaps in this knowledge can then be supplemented by live traversal prior to or during execution

The challenge is determining when a cache is "complete" for the current query, as missing or stale data can affect the correctness of identified exclusive groups. For example, if the cached data originates from a different query, the cached traversal may differ because URI reachability [31] depends on the current query's content. Consequently, the cached view may omit URIs that were unreachable during the previous query.

Authoritativeness Assumptions: Establishing exclusive groups in a decentralized dataspace could be achieved through the notion of *source authority* [33]. Under standard Link Traversal, any reachable source can assert triples about any subject (e.g., a third-party source asserting `<P1> foaf:knows <P2>`), creating a "wild west" of data where no single source can be deemed the exclusive authority for a topic. This lack of authority prevents the query engine from determining if a set of patterns can be safely delegated to a single dataspace or endpoint.

RT: I'd position this around our work on guided link traversal, where adding additional assumptions on the link structures can help us optimize more. A possible solution is restricting the scope of validity such that a data source is the *sole authority* for triples where the subject corresponds to the source's own URI (or a URI within its controlled namespace). By enforcing this *Subject Authority Constraint*, the query engine can statically determine that all triples matching `<SourceA/posts/post1> ?p ?o` reside exclusively at *Source A* [1]. This allows the engine to safely treat such patterns as an exclusive group, enabling the delegation of complex sub-queries (e.g., BGPs) to hybrid sources like SPARQL endpoints without the risk of incomplete results.

3.2. Integrating Dynamically Delegated Sub-queries into the Query Plan

In the traditional optimize-then-execute approach, used by Comunica-link-traversal [27, 28] and SQUIN [32] (except in [34]), the query plan is constructed before execution and evaluated over dynamically discovered sources.

However, exclusive groups are only identified during query execution, when expressive interfaces such as SPARQL endpoints are discovered through link traversal.

Because the execution plan is already fixed, integrating an exclusive group from a newly discovered interface requires dynamically transforming the plan to accommodate the subquery.

Example 3.1. As an example, consider a query plan with the join structure

$$(T_1 \bowtie T_2) \bowtie (T_3 \bowtie T_4).$$

If the engine discovers an exclusive group consisting of the triple patterns $\{T_2, T_3\}$, this group cannot be integrated into the plan. The plan never produces an intermediate result of the form $T_2 \bowtie T_3$, since these patterns occur in different subtrees. As a result, the exclusive group does not fit into the existing join structure and cannot be pushed as a single subquery. However, if the plan is reordered to

$$(T_2 \bowtie T_3) \bowtie (T_1 \bowtie T_4),$$

this group can be integrated, allowing the sub-query $\{T_2, T_3\}$ to be sent to the appropriate data sources.

3.2.1. Research Avenues

Adaptive query processing offers a clear path forward. It enables low-cost plan switching without discarding intermediate results, preventing plan thrashing caused by the large number of sources in a dataspace. Algorithms such as Eddies [35], SteMs [36] (used in [34]), and STAIRs [37] are viable candidates due to their tuple-level adaptivity.

However, the literature lacks a comprehensive analysis of these adaptive frameworks within LTQP. Specifically, there is little guidance on designing routing strategies, the logic for forwarding tuples between operators, and assessing their overhead costs.

Despite this gap, two works are relevant. The first [34] uses tuple-level routing combined with adaptive URI prioritization. However, they evaluate the performance of the URI prioritization and do not present the performance difference of using tuple-level routing instead of a fixed query plan. As prioritization performs poorly in Solid dataspaces [38], these findings lack direct relevance. The second approach [39] uses join restarts to improve query performance. However, if two sources admit incompatible exclusive groups, restarting the join cannot accommodate both.

Given the limitations of these prior works, future research should investigate the use of tuple-level adaptive query processing techniques.

Eddies offer a constrained form of adaptivity. In this model, the join operators are instantiated at the start of the query; the engine only adapts the *order* in which tuples visit these operators. Because these join operators (e.g., Symmetric Hash Joins) accumulate internal state as they process data, the engine is effectively locked into the existing operator instances.

Example 3.2. Consider a query with triple patterns $\{T_1, T_2, T_3, T_4\}$. If an Eddy starts by routing results from $T_1 \bowtie T_2$, the join state is physically stored within that specific operator's hash tables. If the engine later discovers that $T_2 \bowtie T_3$ is more selective due to newly traversed data, it can change the routing sequence, but it cannot easily dissolve the $T_1 \bowtie T_2$ state to form a $T_2 \bowtie T_3$ join without significant re-initialization overhead.

[1]This includes fragments of the URI

In contrast, **SteMs** (State Modules) decouple state from the join operation entirely. Since SteMs do not hold fixed join states, any tuple can be routed to any SteM in any order. However, this flexibility introduces a *recomputation tax*: if a SteM is probed before its join partners have arrived, the intermediate result is not captured and must be re-derived once more data is discovered via traversal.

Finally, **STAIRs** (Storage, Transformation and Access for Intermediate Results) attempt to bridge this gap through state migration. This allows a query engine to physically move join states between different operator instances. While this maximizes flexibility, state migration is a resource-intensive operation. Determining the optimal threshold for migration, and the number of required migrations to incorporate the diverse precomputed sub-queries discovered during traversal is an open challenge.

3.3. Service Discovery in Hybrid Query Environments

A fundamental challenge in Hybrid Link Traversal Querying is the gap between resource identifiers in Linked Data documents and the aggregated query services that host them. For standard LTQP, the reachability of data is determined by the recursive dereferencing of URIs, adhering to the "follow-your-nose" principle where a URI identifies a resource and provides its description. However, SPARQL endpoints function as aggregators: they contain data about resources but do not necessarily share the resource's URI namespace, nor are they always explicitly linked from the resource descriptions themselves. Consequently, a query engine may possess a URI for resource A, even though the resource itself is only accessible via a SPARQL endpoint at a different URI. Because of this decoupling, the engine cannot rely on the "follow-your-nose" principle to dereference resource A directly; it must discover the relevant endpoint instead. For example, an engine might encounter the URI `https://alice.example/posts/1` representing a social media post. Dereferencing this URI directly yields no RDF data. Instead, all data regarding Alice's posts resides exclusively in a SPARQL endpoint located at a different URI, such as `https://alice.example.org/posts/sparql`. Because the resource and endpoint URIs are decoupled, standard link traversal fails to retrieve the data.

3.3.1. Research Avenues

Explicit Linkage Mechanisms An avenue for mitigating the service discovery gap is by using explicit links between resources and their hosting endpoints. This approach relies on extending the "follow-your-nose" principle to include service metadata within the resource description itself. Existing methods such as the Vocabulary of Interlinked Datasets (VoID) can embed pointers directly in the RDF data. For instance, a triple taking the form $\langle r, \texttt{void:sparqlEndpoint}, s\rangle$ allows a traversal engine to immediately identify that the resource r is queryable via the endpoint s. However, this relies on adoption of this approach while, for example, currently only 33% percent of endpoints expose VoID descriptions [40]. Furthermore, generating individual triples for every term may be infeasible. Methods should therefore be investigated to simplify these statements, such as using regex patterns to define what types of resources are constrained in the endpoint.

Service Registries and Indexing. Instead of relying on direct linkage, research can be directed towards the usage of external service registries and catalogs. Unlike linkage-based discovery, this approach treats the mapping between URI namespaces and SPARQL endpoints as a separate knowledge base. Within a dataspace, services such as ESPRESSO [25] can aggregate these connections, effectively bridging the gap between identifiers and query services. However, relying on purely centralized indexes is antithetical to the goals of a decentralized dataspace. Consequently, future directions should investigate distributed alternatives, such as Peer-to-Peer (P2P) discovery mechanisms, to prevent the reliance on single points of failure.

3.4. Alternative View Deduplication

In decentralized dataspaces, a single dataset can often be accessed through multiple architectural views. Consider a personal data store (PDS) following the Solid protocol [41]. Based on the Linked Data

Platform (LDP) [42], this environment structures data as a hierarchical resource tree, interconnecting documents via RDF predicates.

Data providers may also expose a SPARQL endpoint over the same PDS to improve query efficiency. Ideally, an LTQP engine should incorporate this endpoint into its query plan. However, because endpoint discovery is itself a byproduct of the traversal process, an engine may dereference several hypermedia documents before it encounters the metadata pointing to the endpoint.

This creates a synchronization problem: the engine ingests the same data from two different interfaces, leading to duplicated results and wasted computational resources. In the specific context of Solid, a client might assume that any (public) URI sharing the Pod's root namespace is covered by the discovered endpoint. However, in general dataspace environments, this assumption is often unsubstantiated; alternative views may only cover specific subsets of data, such as historical archives versus high-frequency data. In addition, views may cover data outside the namespace, if it acts as a aggregator, further complicating the problem of determining which sources will cause result duplication.

3.4.1. Research Avenues

View Overlap Detection To determine an appropriate deduplication strategy, a query engine must first assess the data coverage of the discovered access interfaces. One approach from the literature involves issuing queries to both the local data store and the remote endpoint to calculate *coherence* [19], a metric indicating the extent to which the two sources agree. However, this method is restricted to queryable interfaces and incurs significant overhead, as each coherence check requires additional HTTP requests. An alternative is to rely on server-side metadata that explicitly denotes which underlying resources a given access interface aggregates. Unfortunately, this relies heavily on the widespread adoption of specific metadata vocabularies by data providers, which cannot easily be guaranteed in a decentralized, autonomous dataspace.

Client-Side Deduplication Regardless of whether the data coverage of discovered access interfaces is known, the query engine must perform client-side deduplication. This requires tracking the *provenance* (i.e., the source) of intermediate results to filter out duplicates.

We consider two scenarios based on this coverage knowledge. First, without coverage guarantees, the engine relies on a naive approach. It maintains a "seen" set of all previously produced intermediate results for each triple pattern, discarding any new matches that already exist in the set. Retaining this complete history is highly memory-intensive and scales poorly. While probabilistic data structures like Bloom filters can reduce this memory footprint, their inherent false-positive rates risk accidentally discarding valid results.

Second, if the engine knows which sources an alternative access interface covers, it can optimize the lifecycle of the "seen" sets. Once the engine guarantees that all data capable of producing duplicate results has been processed, it can safely discard the associated sets to free memory and prevent sources covered by the alternative interface from being dereferenced. However, reliably detecting this execution point remains an open research problem.

4. Conclusion

Hybrid querying over heterogeneous sources has the potential to significantly reduce query latency and network overhead, by combining the dynamic discovery of link traversal with the computational efficiency of expressive interfaces like SPARQL endpoints,

However, as outlined in this paper, realizing a fully functional Hybrid LTQP engine introduces substantial technical hurdles. The unbounded nature of the web makes it difficult to establish exclusive groups for safe sub-query delegation without relying on prior knowledge, discoverable indexes, or authoritativeness assumptions. Furthermore, query engines must adopt highly adaptive processing techniques to integrate dynamically discovered materialized sub-queries without thrashing the query plan or losing intermediate results. We also highlight the ongoing need for robust service discovery mechanisms to bridge the gap between resource identifiers and the endpoints aggregating them. Finally, we identify a challenge with alternative views and the inherent risk of result duplication due to them and identify two reseach directions for this challenge. Within these challenges possible solutions are by the inclusion of various metadata regarding the content of data sources within dataspaces, which would be best to take up within standardization efforts.

Ultimately, transitioning from a traversal strategy where only single triple patterns can be sent to expressive interfaces to a fully hybrid approach is essential to leverage the diverse access interface expected in dataspaces. Addressing these open research avenues will be critical in developing query engines capable of scaling to the realistic demands of a decentralized data ecosystem.

Acknowledgements

This research was supported by SolidLab Vlaanderen (Flemish Government, EWI and RRF project VV023/10). Ruben Taelman is a postdoctoral researcher at the Research Foundation – Flanders (FWO).

Declaration on Generative AI

During the writing of this paper, the author(s) used Gemini in order to: Sentence Polishing, Rephrasing, and Text Creation. 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] A. Schwarte, P. Haase, K. Hose, R. Schenkel, M. Schmidt, Fedx: Optimization techniques for federated query processing on linked data, in: The Semantic Web–ISWC 2011: 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I 10, Springer, 2011, pp. 601–616.

[2] M. Schmidt, O. Görlitz, P. Haase, G. Ladwig, A. Schwarte, T. Tran, Fedbench: A benchmark suite for federated semantic data query processing, in: The Semantic Web–ISWC 2011: 10th International Semantic Web Conference, Bonn, Germany, October 23-27, 2011, Proceedings, Part I 10, Springer, 2011, pp. 585–600.

[3] L. Heling, M. Acosta, Federated sparql query processing over heterogeneous linked data fragments, in: Proceedings of the ACM Web Conference 2022, 2022, pp. 1047–1057.

[4] M.-H. Dang, J. Aimonier-Davat, P. Molli, O. Hartig, H. Skaf-Molli, Y. Le Crom, Fedshop: a benchmark for testing the scalability of sparql federation engines, in: International Semantic Web Conference, Springer, 2023, pp. 285–301.

[5] L. Heling, M. Acosta, A framework for federated sparql query processing over heterogeneous linked data fragments, arXiv preprint arXiv:2102.03269 (2021).

[6] S. Cheng, O. Hartig, Towards query processing over heterogeneous federations of rdf data sources, in: European Semantic Web Conference, Springer, 2022, pp. 57–62.

[7] R. Verborgh, M. Vander Sande, O. Hartig, J. Van Herwegen, L. De Vocht, B. De Meester, G. Haesendonck, P. Colpaert, Triple pattern fragments: a low-cost knowledge graph interface for the web, Journal of Web Semantics 37 (2016) 184–206.

[8] O. Hartig, C. Buil-Aranda, Bindings-restricted triple pattern fragments, in: OTM Confederated International Conferences" On the Move to Meaningful Internet Systems", Springer, 2016, pp. 762–779.

[9] C. Aebeloe, I. Keles, G. Montoya, K. Hose, Star pattern fragments: Accessing knowledge graphs through star patterns, arXiv preprint arXiv:2002.09172 (2020).

[10] T. Minier, H. Skaf-Molli, P. Molli, Sage: Web preemption for public sparql query services, in: The World Wide Web Conference, 2019, pp. 1268–1278.

[11] A. Azzam, J. D. Fernández, M. Acosta, M. Beno, A. Polleres, Smart-kg: Hybrid shipping for sparql querying on the web, in: Proceedings of the Web Conference 2020, 2020, pp. 984–994.
[12] A. Azzam, C. Aebeloe, G. Montoya, I. Keles, A. Polleres, K. Hose, Wisekg: Balanced access to web knowledge graphs, in: Proceedings of the Web Conference 2021, 2021, pp. 1422–1434.
[13] B. Esteves, H. J. Pandit, V. Rodríguez-Doncel, Odrl profile for expressing consent through granular access control policies in solid, in: 2021 IEEE European Symposium on Security and Privacy Workshops (EuroS&PW), IEEE, 2021, pp. 298–306.
[14] A. Padia, T. Finin, A. Joshi, et al., Attribute-based fine grained access control for triple stores, in: 3rd Society, Privacy and the Semantic Web-Policy and Technology workshop, 14th International Semantic Web Conference, 2015.
[15] A. Halevy, M. Franklin, D. Maier, Principles of dataspace systems, in: Proceedings of the twenty-fifth ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems, 2006, pp. 1–9.
[16] E. Curry, Dataspaces: fundamentals, principles, and techniques, in: Real-time Linked Dataspaces: Enabling Data Ecosystems for Intelligent Systems, Springer, 2019, pp. 45–62.
[17] J. Van Herwegen, R. Verborgh, Granular access to policy-governed linked data via partial server-side query, in: European Semantic Web Conference, Springer, 2024, pp. 331–335.
[18] O. Hartig, A. Langegger, A database perspective on consuming linked data on the web, Datenbank-Spektrum 10 (2010) 57–66.
[19] J. Umbrich, A hybrid framework for querying linked data dynamically (2012).
[20] S. J. Lynden, I. Kojima, A. Matono, A. Nakamura, M. Yui, A hybrid approach to linked data query processing with time constraints., LDOW 996 (2013) 1–10.
[21] O. Hartig, K. Hose, J. F. Sequeda, Linked data management., 2019.
[22] G. Ladwig, T. Tran, Linked data query processing strategies, in: International Semantic Web Conference, Springer, 2010, pp. 453–469.
[23] M. M. Sabri, A hybrid framework for online execution of linked data queries, in: Proceedings of the 24th International Conference on World Wide Web, 2015, pp. 515–519.
[24] G. Ladwig, T. Tran, Sihjoin: Querying remote and local linked data, in: Extended Semantic Web Conference, Springer, 2011, pp. 139–153.
[25] M. Ragab, Y. Savateev, R. Moosaei, T. Tiropanis, A. Poulovassilis, A. Chapman, G. Roussos, Espresso: a framework for empowering search on decentralized web, in: International Conference on Web Information Systems Engineering, Springer, 2023, pp. 360–375.
[26] S. Speicher, J. Arwe, A. Malhotra, Linked data platform 1.0, 2015. URL: https://www.w3.org/TR/ldp/, w3C Recommendation.
[27] R. Taelman, J. Van Herwegen, M. Vander Sande, R. Verborgh, Comunica: a modular sparql query engine for the web, in: International Semantic Web Conference, Springer, 2018, pp. 239–255.
[28] R. Taelman, R. Verborgh, Link traversal query processing over decentralized environments with structural assumptions, in: International Semantic Web Conference, Springer, 2023, pp. 3–22.
[29] J. Aimonier-Davat, B. Nédelec, M.-H. Dang, P. Molli, H. Skaf-Molli, Fedup: querying large-scale federations of sparql endpoints, in: Proceedings of the ACM Web Conference 2024, 2024, pp. 2315–2324.
[30] M. Saleem, A.-C. Ngonga Ngomo, Hibiscus: Hypergraph-based source selection for sparql endpoint federation, in: European semantic web conference, Springer, 2014, pp. 176–191.
[31] O. Hartig, J.-C. Freytag, Foundations of traversal based query execution over linked data, in: Proceedings of the 23rd ACM conference on Hypertext and social media, 2012, pp. 43–52.
[32] O. Hartig, Squin: a traversal based query execution system for the web of linked data, in: Proceedings of the 2013 ACM SIGMOD International Conference on Management of Data, 2013, pp. 1081–1084.
[33] B. Bogaerts, B. Ketsman, Y. Zeboudj, H. Aamer, R. Taelman, R. Verborgh, Distributed subweb specifications for traversing the web, Theory and Practice of Logic Programming 24 (2024) 394–420.
[34] O. Hartig, M. T. Özsu, Walking without a map: Ranking-based traversal for querying linked data, in: International Semantic Web Conference, Springer, 2016, pp. 305–324.

[35] R. Avnur, J. M. Hellerstein, Eddies: Continuously adaptive query processing, in: Proceedings of the 2000 ACM SIGMOD international conference on Management of data, 2000, pp. 261–272.
[36] V. Raman, A. Deshpande, J. M. Hellerstein, Using state modules for adaptive query processing, in: Proceedings 19th International Conference on Data Engineering (Cat. No. 03CH37405), IEEE, 2003, pp. 353–364.
[37] A. Deshpande, J. M. Hellerstein, Lifting the burden of history from adaptive query processing, in: Proceedings of the Thirtieth international conference on Very large data bases-Volume 30, 2004, pp. 948–959.
[38] R. Eschauzier, R. Taelman, R. Verborgh, Revisiting link prioritization for efficient traversal in structured decentralized environments, in: International Semantic Web Conference, Springer, 2025, pp. 575–593.
[39] J. Hanski, S. Van Braeckel, R. Taelman, R. Verborgh, Link traversal over decentralised environments using restart-based query planning, in: International Conference on Web Engineering, Springer, 2025, pp. 303–311.
[40] C. Buil-Aranda, A. Hogan, J. Umbrich, P.-Y. Vandenbussche, Sparql web-querying infrastructure: ready for action?, in: International Semantic Web Conference, Springer, 2013, pp. 277–293.
[41] A. V. Sambra, E. Mansour, S. Hawke, M. Zereba, N. Greco, A. Ghanem, D. Zagidulin, A. Aboulnaga, T. Berners-Lee, Solid: a platform for decentralized social applications based on linked data, MIT CSAIL & Qatar Computing Research Institute, Tech. Rep. 2016 (2016).
[42] S. Speicher, J. Arwe, A. Malhotra, Linked Data Platform 1.0, W3C Recommendation, W3C, 2015. URL: https://www.w3.org/TR/ldp/.