SGF: SPARQL Updates over Decentralized Knowledge Graphs without Access Path Dependencies

Jitse De Smet [0009-0002-6513-5013], Ruben Taelman [0000-0001-5118-256X]

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

Abstract. Decentralized data ecosystems, such as the Solid project, empower users to control their data but introduce complexities in data storage and retrieval. Current solutions provide mechanisms for describing data structures but lack sufficient guidance for determining where to create or update resources. To address this challenge, we propose the Storage Guiding Framework (SGF), a framework that enables clients to manage RDF resource storage within Solid pods. This paper introduces SGF, detailing the describing structure and how SGF allows clients to treat Solid pods as RDF collections rather than a collection of unstructured HTTP documents. Our findings show that SGV enhances data accessibility by eliminating the access path data-dependency and providing clear storage strategies. This improvement simplifies client-side data management while maintaining flexibility in data organization.

Keywords: Solid, RDF, Data Management, Update Querying

Source Code: https://github.com/jitsedesmet/sgv-update-engine/tree/demo-eswc
Video: https://youtu.be/4xJFaYyCSIg
Canonical Version: https://sgf-demo-eswc-2025.jitsedesmet.be/

1. Introduction

The Solid project [1] aims to create a decentralized personal data ecosystem based on open standards. Within this ecosystem, each user retains control over their data. Users expose their data through a collection of HTTP documents linked together by Linked Data Platform (LDP) [2], forming what is known as a Solid pod. When querying the data in a solid pod, data consumers, like developers and data analysts, are faced with the access path data-dependency [3]. This means they must know which RDF-resources are exposed through which interfaces at what web-address. For example, social media posts could be exposed in a single document at ‘https://example.org/posts’, but it would be equally valid to expose a collection of documents each grouping the posts of a single day exposed at ‘https://example.org/posts/{creation-date}’. As such, clients that want to access, create or modify data need to explicitly know where to look for the data they need.
To resolve the access path data-dependency, consumers could try to discover all resources exposed by traversing links [4]. However, this method does not scale well as the number of resources increases. To mitigate this, mechanisms can be used to describe the relation between RDF-resources and HTTP-documents that might expose

them. The type index [5], shape tree [6], and shape index [7] explicitly define such relationships, reducing the search space to a limited number of HTTP-documents, effectively functioning as indexes[8].
While a lot of research has gone to querying decentralized environments such as Solid, not a lot of research has gone to updating resources in a decentralized environment. Looking at Solid, the existing descriptions do not provide sufficient information to determine where to create or update resources, because a created or updated RDF-resource might match multiple HTTP-documents, leading to several key questions:

1. *Should the RDF-resource be exposed in all matching HTTP-documents, only some, or just one?*
2. *What should be done if no HTTP-document matches?*
3. *What happens if a resource is updated? Can it be modified at all? Should it be relocated? Does the description change?* Many more of these questions can be asked, and to address them,

we introduce the Storage Guidance Vocabulary (SGV) and accompanying Storage Guiding Framework (SGF) [9]. SGV allows the structure of a Solid pod to be described, while SGF uses this description to guide clients in managing RDF-resources within Solid pods. Using SGF, clients can interpret the Solid pod as a collection of RDF-data, instead of a collection of HTTP-documents containing RDF-data.

2. Storage Guiding Framework

The Storage Guidance Vocabulary, used by SGF, provides a structured way to describe the relationships between RDF resources in a Solid pod and the LDP mechanisms that expose them via HTTP documents. This section provides a high level overview of the vocabulary and its use, details are omitted for brevity but can be found in previous work [9].
SGV adopts a collection-based approach, where resources are organized into one or more *resource collections*. We define the following types of collections:

1. *Resource Collection*: A group of related RDF resources.
2. *Unstructured Collection*: A traditional LDP container or HTTP document.
3. *Canonical Collection*: A structured resource collection containing specific resources.

Canonical Collections are further described by the following properties:

1. *Resource Description*: A method to describe resources, such as ShEx [10] or SHACL [11].
2. *Group Strategy*: Describes how resources are grouped (e.g., posts grouped by creation date).
3. *Store Condition*: Describes which collection(s) store a resource when multiple collections are eligible, creating a priority system.
4. *Update Condition*: Specifies behaviour when a resource within a collection is modified.

5. *Client Control*: Defines the extent to which a client can deviate from the SGV description.

Using SGF, **RDF resources can be created** in a Solid pod through the following process:

1. Client fetches the SGV description of the target pod.
2. Client checks all *canonical collections*, filtering out those whose *resource description* does not match the to-be-created resource.
3. Client determines which *canonical collections* will store the resource based on *storage conditions*.
4. For each selected collection, the client applies the *grouping strategy* to compute the appropriate HTTP document to store the RDF-resource.
5. Client performs the necessary HTTP requests.

A similar process is used for **updating and removing RDF resources**:

1. Client fetches the SGV description of the target pod.
2. Client virtually applies the update by computing the triple difference, and fetching required HTTP documents.
3. For each modified RDF resource, the corresponding update condition is evaluated, which might result in: a) a prohibition of the modification, or b) an indication that the resource should be relocated to a new collection.
4. Client performs the necessary HTTP requests.

SGF has been implemented within a SPARQL [12, 13] query engine that wraps around the Comunica SPARQL query engine [14]. This engine enables data consumers to both query and update Solid pod data while interpreting the pod as a collection of RDF data rather than a set of HTTP documents containing RDF-data. This eliminates the access path data-dependency previously faced.

3. Demonstration

The demonstration showcases how the SGF query engine enables querying and updating data in a Solid pod. The demo pods are derived from those used in SolidBench [8] and have been enriched with SGV descriptions. As such, the pods follow the same data model as SolidBench, where each pod contains posts and comments from the pod owner. The posts and comments are exposed through four different fragmentation strategies, namely: 1. grouped by creation date, 2. grouped by location, 3. each post and comment has its own HTTP-document, and 4. all posts and comments are grouped together.

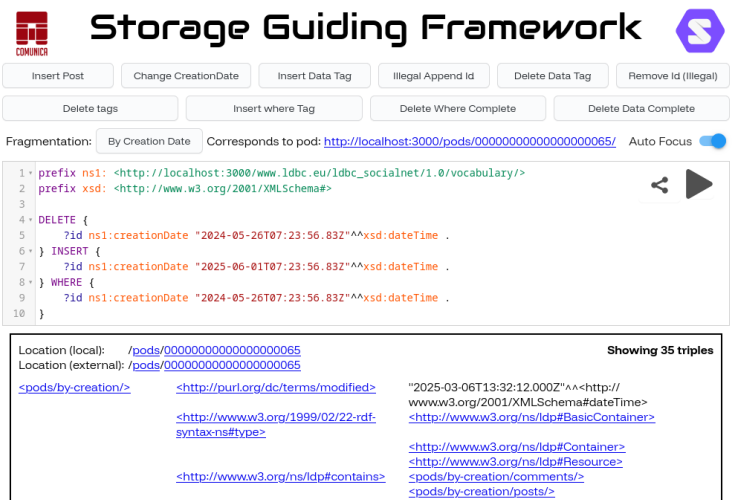

Fig. 1: UI of demo. **Source Code + Docker** *(https://github.com/jitsedesmet/sgv-update-engine/blob/demo-eswc/README.md)*, **Video** *(https://youtu.be/4xJFaYyCSIg)*.

Users can load prepared queries into the query editor by selecting them via buttons at the top of the web-interface or write their own queries. Executing the query in the query editor will perform the required updates in the pod selected using the fragmentation selector. A simple triple browser has been implemented allowing you to browse the different pods using the web interface. After executing a query, the triple browser will automatically reload the focussed HTTP-document and highlight added triples.
When the *Auto Focus* switch is enabled, the triple browser will automatically focus an HTTP-document that was modified by running the query. Additionally, a direct link to the focused HTTP resource is provided for easy navigation to the pod's web interface.

4. Conclusion

In this work, we introduced a framework that enables smart clients to autonomously determine the appropriate HTTP document(s) for storing newly created or updated data on an LDP interface. Our work includes mechanisms to verify whether a resource can be created or deleted. We validated the expressiveness of our descriptive vocabulary, SGV, by implementing an SGF SPARQL engine. Essentially, SGV provides structure to an otherwise unstructured document store based on LDP, by defining a server-side description of the structure that clients should follow. However, since SGV adopts a collection-based approach, instead of a data-centric approach, the application is limited to collection based interfaces, and future research should focus on data descriptions that relieve the data access path dependency in a more general way.

Acknowledgements. This research was supported by SolidLab Vlaanderen (Flemish Government, EWI and RRF project VV023/10). Jitse De Smet is a predoctoral fellow of the Research Foundation – Flanders (FWO) (1SB8525N). Ruben Taelman is a postdoctoral fellow of the Research Foundation – Flanders (FWO) (1202124N).

References

1. Verborgh, R.: Re-Decentralizing the Web, For Good This Time. In: Seneviratne, O. and Hendler, J. (eds.) Linking the World's Information: Essays on Tim Berners-Lee's Invention of the World Wide Web. pp. 215–230. ACM (2023). doi:10.1145/3591366.3591385
2. Speicher, S., Arwe, J., Malhotra, A.: Linked Data Platform 1.0. W3C, https://www.w3.org/TR/2015/REC-ldp-20150226/ (2015).
3. Codd, E.F.: A relational model of data for large shared data banks. Communications of the ACM. 13, 377–387 (1970).
4. Hartig, O.: Zero-knowledge query planning for an iterator implementation of link traversal based query execution. In: Extended Semantic Web Conference. pp. 154–169. Springer (2011).
5. Turdean, T., Zucker, J., Balseiro, V., Capadisli, S., Berners-Lee, T.: Type Indexes. https://solid.github.io/type-indexes/ (2022).
6. Prud'hommeaux, E., Bingham, J.: Shape Trees Specification. https://shape-trees.org/TR/specification/ (2021).
7. Tam, B.-E., Taelman, R., Colpaert, P., Verborgh, R.: Opportunities for Shape-based Optimization of Link Traversal Queries. arXiv preprint arXiv:2407.00998. (2024).
8. Taelman, R., Verborgh, R.: Link Traversal Query Processing over Decentralized Environments with Structural Assumptions. Proceedings of the 22nd International Semantic Web Conference. (2023).
9. De Smet, J.: Abstracting Data Updates over a Document-oriented interface of a Permissioned Decentralized Environment. http://lib.ugent.be/catalog/rug01:003215161 (2024).
10. Baker, T., Prud'hommeaux, E.: Shape Expressions (ShEx) 2.1 Primer. https://shex.io/shex-primer-20191009/ (2019).
11. Knublauch, H., Kontokostas, D.: Shapes Constraint Language (SHACL). W3C, https://www.w3.org/TR/2017/REC-shacl-20170720/ (2017).
12. Seaborne, A., Harris, S.: SPARQL 1.1 Query Language. W3C, https://www.w3.org/TR/2013/REC-sparql11-query-20130321/ (2013).
13. Gearon, P., Passant, A., Polleres, A.: SPARQL 1.1 Update. W3C, https://www.w3.org/TR/2013/REC-sparql11-update-20130321/ (2013).
14. Taelman, R., Van Herwegen, J., Vander Sande, M., Verborgh, R.: Comunica: A Modular SPARQL Query Engine for the Web. In: Proceedings of the 17th International Semantic Web Conference. pp. 239–255. Springer International Publishing (2018). doi:10.1007/978-3-030-00668-6_15