Linked Data Event Streams in Solid LDP containers

Wout Slabbinck[1,*], Ruben Dedecker[1], Sindhu Vasireddy[1], Ruben Verborgh[1] and Pieter Colpaert[1]

[1]IDLab, Departement of Electronics and Information Systems, Ghent University - imec, Belgium

Abstract

The Solid Project – at the time of writing – uses containers with resources in them as defined in the LDP specification as a way to give developers the flexibility to write to a storage in the way they see fit. With cross-app interoperability and read performance in mind, choosing an application profile and container-resource structure becomes guesswork for the app writing the data, as all possible apps reading from the storage are not yet defined. Event sourcing is a technique used in data architecture to decouple writing from reading. Multiple views will always stay in-sync with an event source, or allow one to view a historic state or study the changes that happened over time. In this paper, we study whether we can use the current version of the Solid protocol to store an event source using the Linked Data Event Streams (LDES) specification. We successfully implemented a client library, which we tested on the use case of storing your live location with history, for both reading and writing in two modes: version aware and version agnostic. However, the current Solid protocol based on LDP also shows some limitations towards event sourcing: (i) re-balancing the hypermedia structure publishing the LDES is not possible due to slash semantics, (ii) as the event source is fully managed by clients, a faulty client may corrupt the event source, and (iii) the client is also in charge of enforcing the retention policy, having to delete older resources one by one, while they have no information about the internal limits of the Solid storage. We conclude that the Solid spec as-is can be used to store an event source, and that client libraries can create an abstraction of the history without any server-specific functionality. However, we also had to work our way around some limitations, putting more strain on the client, and want to open the discussion on whether the Solid server protocol needs to be extended for more native support of the event sourcing pattern.

Keywords

Solid, LDP, LDES, Event Sourcing

1. Introduction

In the Solid Project[1], app developers decide the strategy to write – and thus also read – data within the Solid ecosystem. Leaving those decisions to the developer that first writes the

Managing the Evolution and Preservation of the Data Web (MEPDaW 2022)

*Corresponding author.

wout.slabbinck@ugent.be (W. Slabbinck); ruben.dedecker@ugent.be (R. Dedecker); sindhu.vasireddy@ugent.be (S. Vasireddy); ruben.verborgh@ugent.be (R. Verborgh); pieter.colpaert@ugent.be (P. Colpaert)

https://rubendedecker.be/ (R. Dedecker); https://ruben.verborgh.org/ (R. Verborgh); https://pietercolpaert.be/ (P. Colpaert)

0000-0002-3287-7312 (W. Slabbinck); 0000-0002-3257-3394 (R. Dedecker); 0000-0002-3522-5504 (S. Vasireddy); 0000-0002-8596-222X (R. Verborgh); 0000-0001-6917-2167 (P. Colpaert)

CEUR Workshop Proceedings (CEUR-WS.org)

[1]https://solidproject.org/

data (e.g., using footprints[2] or by planting shape trees[3]) raises the read efficiency for only a few use cases, and limits it for others. For example, if an app writing my live location writes each update in a file `location/{timestamp}.ttl`, then we can imagine an app – even if it uses exactly the same application profile – that wants to check when I was last in Paris will take a long time, whereas a write strategy with a geospatial sub-structure such as `location/{geohash}/{timestamp.ttl}` would work faster if this structure is transparent to the client. The application writing is thus in some way biased towards a specific type of reuse, and cross-app interoperability is hampered.

Since cross-app interoperability is one of the main ideas behind the Solid project, we want to introduce a generic write-strategy in which any other kind of "view" (historic or live) on top of this data can be built. A `view` can stay in-sync with the event source and translate the data to something that can then be used specifically by the application at hand. In data architecture, the strategy to build up a history, and keep derived view in-sync with the source is called the **event sourcing pattern** [1]. In event sourcing, each change by an application results in appending an immutable event to an event log.

In this paper, we study how we can support event sourcing within Solid without changing the current server specification. We (i) introduce a specification to write an event source to a Solid storage following the current specification, (ii) create an implementation of a client library to facilitate the management of the event source, and (iii) demo a live location sharing application following this specification. In the conclusion, we then discuss the limitations of this approach.

2. Preliminaries

The **Solid protocol**[4] is a web standard designed to give people control over their online data. The specification is built with existing W3C standards that together form a basis for decentralized data storage.

An agent, such as an end-user, can interact with a Solid storage through create, read, update, and delete (CRUD) operations using the **Linked Data Platform (LDP)** API.[5] Each operation interacts with an LDP Resource. A special kind of resource is the LDP Container, which represents a collection of resources. Creating hierarchical structures within a storage is achieved using containers. This is what we will further refer to as **the container-resource pattern**.

An agent is identifiable with a WebID, which is an HTTP IRI. Derefencing the WebID leads to a profile document where various aspects of the agent are stored. Solid servers can authenticate agents by following the Solid OpenID Connect (Solid-OIDC) specification.[6] Authorization over resources in a storage is defined by the Web Access Control (WAC) specification.[7] It defines how to give specific permissions on resources to agents with Access Control Lists (ACLs).

The **Linked Data Event Streams (LDES)** specification[8] defines an immutable collection of

[2] https://www.w3.org/DesignIssues/Footprints.html
[3] https://shapetrees.org/
[4] https://solidproject.org/TR/protocol
[5] https://www.w3.org/TR/ldp/
[6] https://solidproject.org/TR/oidc
[7] https://solidproject.org/TR/wac
[8] https://w3id.org/ldes/specification

members and is an extension of the TREE hypermedia specification.[9] TREE defines hypermedia controls to navigate over a large collection of members. LDES is designed to be an append-only publishing interface [2] which indicates that it is an Event Source. A **versioned LDES** is a type of event stream that deals with versioned members. Each versioned member has a timestamp and a link to the entity of which it is a version.

3. Related Work

Meinhardt et al [3] discussed five different reasons or use cases to adopt a version-based approach for entities in knowledge graphs: (i) Version References and Data Consistency, (ii) Change Inspection, (iii) Data Quality Assessment, (iv) Dynamic Processes, and (v) Data Dynamics. The fact that the authors provide these arguments indicates that to allow as many use cases to build on data from a Solid pod, it is required that the data is stored in a structure that is version-based.

The Memento framework[10] can be used for DateTime negotiation to interact with all the versions in the event source. Fedora[11] and Trellis[12] [4, 5] show how Memento can work in combination with LDP. With this protocol, clients can negotiate the DateTime of resources, through HTTP requests, with those enhanced LDP servers. Server-side support for the event source is thus required in order to interact with different versions with Memento.

As a running example in this paper, we will use writing your live location to a Solid pod. Van de Winkel et al [6] already created a Solid location app that works with multiple versions of the location points. However, they update the same LDP resource to store new versions, which is a custom implementation that this way will only work for their app. Our challenge is to generally solve this problem for all use cases with live data, from slow-moving to fast-moving.

4. Event Sourcing in Solid with the LDES in LDP specification

Solid is currently built on the container-resource pattern of the LDP protocol without Memento or Fedora. In this chapter, we assume we cannot influence the server specification, and thus want to find a specification to describe an Event Source within this container-resource pattern. We identified a versioned LDES as a perfect match as it does not per se require any changes to the server specification:

1. LDES is an RDFS vocabulary that allows to describe a collection of immutable members. Each member can also be described as a versioned entity if they indicate this through a `dct:isVersionOf` property (or something similar).
2. LDES uses TREE to fragment itself into a materializable interface. An interface that is materializable can be stored into basic files, which thus becomes straightforward for usage on top of a read-write interface that employs the container-resource pattern.

[9] https://w3id.org/tree/specification
[10] https://rfc-editor.org/rfc/rfc7089.txt
[11] https://fedora.info/2018/11/22/spec/
[12] https://www.trellisldp.org/

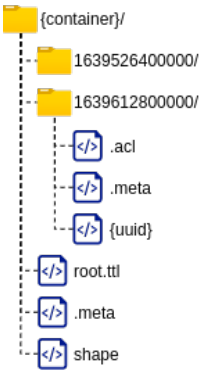

Figure 1: The strategy used for storing a Linked Data Event Stream in an LDP container-resource structure.

In order to write an LDES into an LDP container-resource structure, we introduce the **LDES in LDP specification**[13]. While the up-to-date specification can be fully read on its webpage, we discuss a couple of highlights here.

In Figure 1, the container-resource structure is provided. Data is always being POST-ed to a specific container until it is full. This container is indicated by the root container's `ldp:inbox` property. When the container is full, the client needs to create a new container, start writing there, and update the `ldp:inbox` property of the root container. Furthermore, the client will also keep the `root.ttl` file in-sync with all sub-containers, now describing the event source. This event source contains links to all containers, on which also TREE metadata is added through the `.meta` resource.

5. Client implementation

An implementation of the LDES in LDP specification was made and applied to the use case of storing and reading location data to an Event Source.

In the following subsections, we elaborate on an implementation of a client library of the LDES in LDP specification. Next, we explain how location data can be stored and retrieved with the demo application.

5.1. Implementation of LDES in LDP specification

We published *@treecg/versionawareldesinldp*[14] on npm. This client library allows end-users to both interact with an LDES in LDP as with a versioned LDES in LDP.

5.1.1. LDES in LDP Protocol

The first component of the library interacts with the LDES in LDP Protocol as is. For this the class *LDESinLDP* is used for **initializing** an LDES in LDP, **appending** a member and **creating**

[13] https://woutslabbinck.github.io/LDESinLDP/
[14] https://doi.org/10.5281/zenodo.7234355

a new fragment.

The initialization method executes the following operations in order: (i) creates the root container, (ii) stores the LDES and first view information, (iii) creates the first fragment and (iv) adds the `ldp:inbox` property to the root container.

Adding data to the LDES in LDP is done with the *append* method. First, it retrieves the write location, through extracting the inbox URL from the root container. This inbox URL is then used to send an *HTTP POST* request with as body the data that an end-user wants to be appended to the LDES in LDP.

When a container is full, the *newFragment* method must be executed. It takes care of the creation of a new container, updates the `ldp:inbox` property and finally adds a new `tree:Relation` to the view to keep `root.ttl` in sync.

5.1.2. Versioned LDES in LDP Protocol

The second component of the library provides the tools to interact with the LDES in LDP in a version-based manner. With a version-based approach, the component has to deal with both the object identifier and the timestamp of the **version-objects**. In listing 1, an example of a versioned LDES is given with one version-object. The object identifier, marked by `dct:isVersionOf`, is `ex:resource` and the timestamp, marked by `dct:issued` is `2021-12-15T10:00:00.000Z`.

Listing 1: Example of a versioned LDES with one version-object.

```
ex:ES a ldes:EventStream;
   ldes:versionOfPath dct:isVersionOf;
   ldes:timestampPath dct:issued;
   tree:member ex:resource1v0.

ex:resource1v0
   dct:isVersionOf ex:resource1;
   dct:issued "2021-12-15T10:00:00.000Z"^^xsd:dateTime;
   dct:title "First version of the title".
```

The class *VersionAwareLDESinLDP* abstracts the LDES in LDP protocol by providing CRUD methods to interact with version-objects. When reading an object, a snapshot is taken over all version-objects and the most recent version of the object is returned. When a timestamp t is provided, the result of the snapshot is the most recent version of the object until time t. Additionally, when the client want to provide a version-agnostic view to the developer, the read method can return a materialization of the version-object.

Creating, updating and deleting a member is done by appending a version-object to the LDES in LDP together with a timestamp to indicate when this action happened. Furthermore, there is the *extractVersions* method that extracts all the versions for a given object identifier. Optionally, this can be constrained by a time window.

5.2. Live location demo

To verify the implementation of the protocol, we applied it to the use case of live location data and created two applications:

- A Solid application with a User Interface that tracks the location of a user in real-time and has the functionality to share this location with another user and vice versa.[15]
- A command-line interface (CLI) service that can transform location points and store them in an event source in Solid.[16]

In this demo, we take into account multiple ways of generating location data. We have created a service that can digest those different inputs and append the measurements into the Solid event source.

In the first approach, the Location History application is used to track your location and store them in a container using the model described in the previous subsection. The service reads this container and follows the LDES in LDP specification to store the measurements in the event source.

A second approach uses data generated from third-party applications which can be exported to the GPX file format.[17] The service supports adding track points from a GPX file to the event source. As GPX is not RDF, there first needs to be a transformation from the points to the model, which we accomplish with the RDF Mapping Language (RML) [7].[18] After the mapping, the measurements are appended to the event source.

6. Conclusion

We conclude that writing an event source within Solid LDP containers is possible thanks to the Linked Data Event Streams vocabulary and the TREE specification. However, we have stumbled on three limitations when we use the Solid protocol on the server as-is:

1. In the LDES in LDP specification, a B+ Tree structure is introduced as a way to optimize the LDES to deal with a large number of resources. An important aspect of having a B+ Tree is the fact that it can increase or decrease in size easily through **re-balancing**. However, in LDP, re-balancing would require moving resources from one container to the other, but that would entail their URIs need to change due to **slash semantics**[19]. This hinders the long-term scalability of the event source.
2. **Clients** are required to write to an **LDES in LDP by strictly following the protocol**. However, when there is a **bug** in one of the clients that write to the storage, it may **corrupt the event source** for everyone. As an illustration, there could be a client that

[15] https://doi.org/10.5281/zenodo.7234420
[16] https://doi.org/10.5281/zenodo.7234316
[17] https://wiki.openstreetmap.org/wiki/GPX
[18] https://rml.io/specs/rml/
[19] https://solidproject.org/TR/protocol#uri-slash-semantics

does not update the `ldp:inbox` property with the consequence that new members are added in the wrong fragment. The LDES view is now invalid, which leads to clients using the TREE hypermedia relations not being able to retrieve the correct objects.

3. **Retention policies** on top of a `tree:ViewDescription` of an LDES make sure that the storage requirements are controlled. With a client fully organizing and maintaining the event source, the client will also be in charge of enforcing the retention policy. This means that it needs to run regularly to actively delete older resources one by one. While it is not desirable for the client for efficiency reasons, it is also not desirable for the server, as probably it is the server that will need to indicate these policies based on its physical storage capacity.

However, if cross-app interoperability (cfr. the introduction) is indeed core to the ideas in the Solid Project, then we express our opinion that event sourcing should be in the core of the Solid specification. We propose to open a discussion within the Solid specification for a new kind of append-only container fully managed by the server. This would allow the server to: (i) create a balanced search tree, and possibly apply slow storage techniques for historic data; (ii) let multiple apps write to a single inbox, and let the server organize it into one or multiple read-views; and (iii) let the server enforce a negotiable retention policy based on application needs and internal storage capabilities. Furthermore, it would allow the server to also support a version agnostic view on the data when developers do not want to be bothered with the complexities of working with version objects.

In future work, we will prototype such a container in the Solid LDES server[20] and propose the design to be added to the Solid protocol.

Acknowledgements

Supported by SolidLab Vlaanderen (Flemish Government, EWI and RRF project VV023/10) and the Flemish Smart Data Space (Flemish Government, Digital Flanders and RRF project VV073).

References

[1] M. Kleppmann, Designing data-intensive applications: the big ideas behind reliable, scalable, and maintainable systems, first edition ed., O'Reilly Media, Boston, 2017. OCLC: ocn893895983.

[2] D. Van Lancker, P. Colpaert, H. Delva, B. Van de Vyvere, J. Rojas Meléndez, R. Dedecker, P. Michiels, R. Buyle, A. De Craene, R. Verborgh, Publishing base registries as Linked Data Event Streams, in: M. Brambilla, R. Chbeir, F. Frasincar, I. Manolescu (Eds.), Proceedings of the 21th International Conference on Web Engineering, volume 12840 of *Lecture Notes in Computer Science*, Springer, 2021, pp. 28–36. URL: https://link.springer.com/chapter/10.1007/978-3-030-74296-6_3. doi:10.1007/978-3-030-74296-6_3.

[3] P. Meinhardt, M. Knuth, H. Sack, TailR: a platform for preserving history on the web of data, in: Proceedings of the 11th International Conference on Semantic Systems,

[20] https://github.com/TREEcg/ldes-solid-server

ACM, Vienna Austria, 2015, pp. 57–64. URL: https://dl.acm.org/doi/10.1145/2814864.2814875. doi:10.1145/2814864.2814875.

[4] G. Jansen, A. Coburn, A. Soroka, R. Marciano, Using Data Partitions and Stateless Servers to Scale Up Fedora Repositories., in: IEEE BigData, 2019, pp. 3098–3102.

[5] G. Jansen, A. Coburn, A. Soroka, W. Thomas, R. Marciano, DRAS-TIC Linked data: Evenly distributing the past, Publications 7 (2019) 50. Publisher: Multidisciplinary Digital Publishing Institute.

[6] M. Van de Wynckel, B. Signer, A Solid-based Architecture for Decentralised Interoperable Location Data, in: 12th International Conference on Indoor Positioning and Indoor Navigation, CEUR Workshop Proceedings, 2022.

[7] A. Dimou, M. Vander Sande, P. Colpaert, R. Verborgh, E. Mannens, R. Van de Walle, RML: A Generic Language for Integrated RDF Mappings of Heterogeneous Data, in: C. Bizer, T. Heath, S. Auer, T. Berners-Lee (Eds.), Proceedings of the 7th Workshop on Linked Data on the Web, volume 1184 of *CEUR Workshop Proceedings*, 2014. URL: http://ceur-ws.org/Vol-1184/ldow2014_paper_01.pdf, iSSN: 1613-0073.