Publishing base registries as Linked Data Event Streams

Dwight Van Lancker[1,3], Pieter Colpaert[1], Harm Delva[1], Brecht Van de Vyvere[1], Julián Rojas Meléndez[1], Ruben Dedecker[1], Philippe Michiels[2], Raf Buyle[1,3], Annelies De Craene[3], and Ruben Verborgh[1]

[1] IDLab, Dep. of Electronics and Information Systems, Ghent University – imec
[2] imec EDiT
[3] Flemish Information Agency

Abstract. Fostering interoperability, Public Sector Bodies (PSBs) maintain datasets that should become queryable as an integrated Knowledge Graph (KG). While some PSBs allow to query a part of the KG on their servers, others favor publishing data dumps allowing the querying to happen on third party servers. As the budget of a PSB to publish their dataset on the Web is finite, PSBs need guidance on what interface to offer first. A core API can be designed that covers the core tasks of Base Registries, which is a well-defined term in Flanders for the management of authoritative datasets. This core API should be the basis on which an ecosystem of data services can be built. In this paper, we introduce the concept of a Linked Data Event Stream (LDES) for datasets like air quality sensors and observations or a registry of officially registered addresses. We show that extra ecosystem requirements can be built on top of the LDES using a generic fragmenter. By using hypermedia for describing the LDES as well as the derived datasets, agents can dynamically discover their best way through the KG, and server administrators can dynamically add or remove functionality based on costs and needs. This way, we allow PSBs to prioritize API functionality based on three tiers: (i) the LDES, (ii) intermediary indexes and (iii) querying interfaces. While the ecosystem will never be feature-complete, based on the market needs, PSBs as well as market players can fill in gaps as requirements evolve.

1 Introduction

Public Sector Bodies (PSBs) world-wide maintain and open up reference datasets to foster interoperability by advocating the reuse of the identifiers for which they are the authoritative source. In Flanders, for example, the Large-scale Reference Database[4] (LRD) contains millions of geospatial objects in the Flemish region [2]. On the one hand, the LRD publishes periodical data dumps or version materializations, which users have to fully download to stay up to date with the dataset.

[4] `https://overheid.vlaanderen.be/en/producten-diensten/large-scale-reference-database-lrd`

With a querying API, on the other hand, users can query the dataset without first having to download the entire dataset. Trying to meet the needs of their reusers, PSBs will have to provide and maintain an increasing amount of such querying APIs as specific end-user features are solved by creating feature-specific APIs [7]. However both data dumps and querying APIs will never fully meet the needs of their end-users, as a data dump gives a possibly outdated view on the dataset, whereas a querying API provides its client only with a partial view of the dataset.

To avoid synchronization problems with data dumps on the one hand, and maintenance problems of an always increasing amount of querying APIs on the other, trade-offs need to be made. This resulted in the question: "**What is the base API for base registries?**". PSBs must accept they will not be able to implement any querying API on their own, but that there are other organizations with other interests that can take up parts of the processing. In Section 2 we discuss the definition of the European term "base registry", the ideas behind Linked Data Fragments and the recent initiative of Streaming Linked Data on which our approach was inspired. In Section 3 we design a Linked Data Event Stream (LDES) incrementally by first making sure everyone can download the history and retrieve the latest updates on the data collection. In Section 4 we then introduce three generic open-source building blocks for a FAIR [10] ecosystem, where also third parties can build reusable indexes on top of an LDES. Finally we discuss in Section 5 the three tiers of base registry management, creating a vision for PSBs to set the priorities when deciding upon their next API.

2 Related Work

The term *base registry* was introduced by the European Commission and is defined as a trusted and authoritative source of information, which can and should be digitally re-used by others. A single organization is responsible and accountable for the collection, use, updating and preservation of information. Authoritative in this context means that a base registry is considered to be the source of information and is thus up-to-date and of the highest quality [5]. In order to publish its base registries for maximum reuse on the Web, the Flemish Information Agency (FIA) embraces Linked Data with the Flemish Interoperability Program called Open Standards for Linked Organizations (OSLO). OSLO develops unambiguous data standards to exchange data in an uniform way [1].

The FIA has aligned its base registries with the definition as stated by the European Commission, but extended it with three additional requirements: (i) Base registries are part of a semantic system of uniform identified objects and relations which are in line with the OSLO standards; (ii) The identifiers of objects in a base registry should be re-used in other base registries (or datasets); and (iii) Each base registry is obliged to have life-cycle and history management of

[5] `http://eurlex.europa.eu/resource.html?uri=cellar:2c2f2554-0faf-11e7-8a35-01aa75ed71a1.0017.02/DOC_1&format=PDF` p. 31–32

their objects [2]. This extended definition is considered to be the *core task* of a base registry.

Today, data controllers publish their data through a querying API, such as the Open Geospatial Consortium (OGC)[6] APIs for example. These APIs build upon the legacy of OGC Web Service standards, of which WFS and WMS are the most known. Although the WFS is a standardised – technical - protocol, it does not provide interoperable data. Data in a smart city is about data in a network, all linked in one way or another, which a WFS service lacks. At the moment, it is even impossible to use a dataset described with the principles of Linked Data as a data source in a WFS service, although recent efforts have been made [3]. Furthermore, the processing done by such a service happens fully on the server side, meaning that all costs are for the provider of it.

Instead of publishing their data through a querying API, data controllers also have the possibility to publish a data dump of the dataset. Both interfaces have in common that they only return a *fragment* of the dataset. Given a Linked Data dataset, the result of each request to such interfaces is called a Linked Data Fragment (LDF)[7]. On the axis of LDFs, both data dumps and querying APIs are situated at the extremes, because the workload needed to compute the fragments is divided differently between clients and servers. In the case of a data dump, the processing burden is put on the client-side, but also allows the most flexibility for the client. In the other case, providing a querying API on top of the dataset puts the processing burden on the server, allowing any kind of query and therefore limiting the availability of the API, i.e. a SPARQL endpoint. In order to achieve efficient Web querying, in-between solutions that provide an optimal balance between client and server effort are needed [8]. In-between solutions exist, such as Triple Patterns Fragments, brTPF, Smart KG and subject pages. These in-between solutions shift the needed processing more towards the client and limit the different queries that can be executed on the server.

Publishing data at a high speed has caused a shift in the data landscape, as such that it does not always make sense anymore to use polling-based approaches. Instead, it makes more sense to push this fast-changing (with an acceptable latency of ≤ 10 seconds), continuously updating dataset to its consumers [9]. In order to manage these streams of data, Stream Processing Engines have come to aid [6]. To counter the problem that a data stream can be in all shapes and sizes, an effort was needed by the Web of Data community. This led to the creation of RDF Stream Processing techniques, which allows to process RDF-based data streams. These ideas were already applied on non-sensor related datasets such as DBPedia and Wikimedia, where the goal was to query of the latest changes, with the term Streaming Linked Data [6]. However, more general the goal should be to provide the ability to query over a window of updates on top of a stream, which is similar to our goal, as we want to provide everyone as fast as possible with the latest updates.

[6] `https://ogcapi.ogc.org/`
[7] `https://linkeddatafragments.org/`

3 A base API for base registries

A Linked Data Event Stream (LDES) extends the principles of an event stream by publishing interoperable data re-using existing machine-readable data standards. We applied this data publishing strategy to two datasets: for context information, we used the registry of all officially registered addresses in Flanders, using the OSLO data standard[8] to describe them. For a faster updating dataset, we used measurements of air quality sensors, using the Semantic Sensor Network Ontology.

```
<C> a tree:Collection  ;
    tree:shape <shacl.shape>  ;
    tree:member <Observation1> .

<Observation1> a sosa:Observation  ;
    sosa:resultTime "2020..."  ;
    sosa:hasSimpleResult "1" .
```

Listing 1.1: Linked Data Event Streams described with the TREE hypermedia API specification

The definition of a LDES is a collection of immutable objects, such as an observation made by a sensor or the version of an address. To describe LDESs, we used a hypermedia specification, called the TREE Hypermedia API specification[9]. Using a hypermedia specification to describe event streams, makes them self-descriptive. There is not really a definition of what an event stream exactly is, which means, in order to replicate the event stream, the links have to be followed. The TREE specification describes a LDES as a `tree:Collection`, containing objects that are immutable, defined as `tree:members`. Each member of the collection has a timestamp that indicates at which time the immutable object was created. Furthermore, with `tree:shape` the specification allows to link a SHACL shape [4] to the collection, indicating the content of the immutable objects. The presence of such a SHACL shape is rather an optimization so that autonomous agents know beforehand what the content of the immutable objects is within the collection. An example of the specification is shown in Listing 1.1 and was also applied to air quality observations: `https://streams.datapiloten.be/observations`.

However when implementing a LDES for data models that do not have the concept of things that live in time, the model must be extended, which is the case for an address or a sensor. It is possible for a sensor to change, take for example the Bel-Air project[10] in Flanders, where air quality sensor were fitted to the roof of mail delivery vans. So periodically, not only the observation made by a sensor changes, but it is also possible that the location of a sensor has changed. The stated problem can be solved by using the concept of versions, for example `dcterms:isVersionOf` can be used, as shown in listing 1.2. The Dutch

[8] `https://data.vlaanderen.be/ns/adres`
[9] `https://treecg.github.io/specification/`
[10] `https://www.imeccityofthings.be/en/projecten/bel-air`

NEN3610 standard[11] for example advocates the use of `foaf:isPrimaryTopicOf`. Given that each sensor is uniquely identifiable, a URI can be created for it. This URI can then be used as the object in the triple with `dcterms:isVersionOf` as predicate. By combining the original URI and the timestamp on which the event was generated, each version object is also uniquely identified, making them individual reusable.

```
<C> a tree:Collection  ;
    tree:shape <shacl.shape>  ;
    tree:member <E1> .

<E1> prov:generatedAtTime "2020-01-01T00:00:00Z"  ;
     adms:versionNote "First version of this address"  ;
     dcterms:isVersionOf <AddressRecord1>  ;
     dcterms:title "Streetname X, ZIP Municipality, Country" .
```

Listing 1.2: When a data model does not have the concept of things that live in time, the model must be extended, for example, with the concept of versions. Here, `dcterms:isVersionOf` is used to indicate which address object is affected by this event.

Furthermore, the TREE Hypermedia API specification was also used to describe the metadata of each page. With `tree:relation`, the specification enables users to describe the relation between a specific value and all members on a page linked from the current page. Using this relation, a query agent can automatically discover whether or not it is useful to go to the next page. The most interesting fragmentation strategy for an event stream is time-based as the data grows in time. As shown in listing 1.3, the first page, which always contains the oldest objects, has a `tree:GreaterThanOrEqualToRelation` with the second page, which indicates that all values of page two are greater than or equal to those of page 1. To indicate on what property of the immutable object the relation is based on, the predicate `tree:path` is used. The predicate `tree:value` then contains the value for which all members on the next page are greater than or equal to. In listing 1.3, `sosa:resultTime` is the property that the relation is based on, and thus all members on `?page=2` have a `sosa:resultTime` that is later than or equal to 2020-12-24T12:00:00Z.

```
<?page=1> a tree:Node  ;
          tree:relation [
              a tree:GreaterThanOrEqualToRelation  ;
              tree:path sosa:resultTime  ;
              tree:node <?page=2>  ;
              tree:value "2020-12-24T12:00:00Z"^^xsd:dateTime .
          ] .
```

Listing 1.3: Within the TREE Hypermedia API specification, a relation to another page can be described with `tree:relation`.

4 A Linked Data Event Streams ecosystem

We implemented three reusable building blocks:

[11] `https://geonovum.github.io/NEN3610-Linkeddata/`

The metadata extractor can be used to read TREE metadata in a page and show the next steps possible from the current page to an app or an intermediary server. The extractor has been written within the Comunica framework [5] and is available at `https://github.com/TREEcg/comunica-feature-tree/`.

The LDES client reuses the metadata extractor to allow intermediary servers to copy all members of a `tree:Collection`, and subscribe to new updates. A fragment's time to live is retained from its HTTP caching headers. A polling interval can be configured to wait before refetching. Specifically for an LDES, before emitting an immutable member of a collection, a cache can be checked to check whether the object has not been emitted before. This way, consumers only retrieve updated members of a collection. Code is available at `https://github.com/brechtvdv/event-stream-client`.

The fragmenter reuses the LDES client to keep its own copy in sync and to refragment the LDES based on a configuration. Code is available at `https://github.com/hdelva/tree_index`.

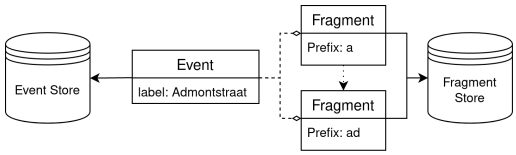

Fig. 1: A schematic overview of the fragmentation process. Values from each individual event are used to place that event into one or more fragments. In this example, the event represents a street labeled as "Admontstraat", and this label is used as the input of a prefix-based fragmenter. The logical links between increasingly specific prefixes are stored in a separate storage layer, which is used to generate the hypermedia descriptions. The events themselves are stored like regular RDF data, and the contents of a fragment are persisted as a set of event URIs.

Applications that require a specific subset of the data can be optimized by consuming only the most relevant data for their use case. For instance, applications that focus on a specific geospatial region are more likely to reuse the published data if they can filter out data from other regions.

To realize this, we have implemented an intermediary server that (re)fragments an existing LDES into multiple smaller ones[12]. Every discovered immutable object is assigned to one or more fragments, as illustrated in Fig. 1. A LDES may be processed using multiple fragmentation strategies, resulting in multiple orthogonal fragmentations, and some strategies can yield multiple

[12] For example, a prefix fragmentation applied to the LDES of streetnames: `https://fast-and-slow.osoc.be/data/streetname/prefix`

fragments for a single event. In the latter case, the fragments can be ordered by increasing specificity such as by prefix length or geospatial granularity. These relations between fragments are stored separately from the events themselves, and are used to generate the hypermedia controls.

5 Conclusion and future work

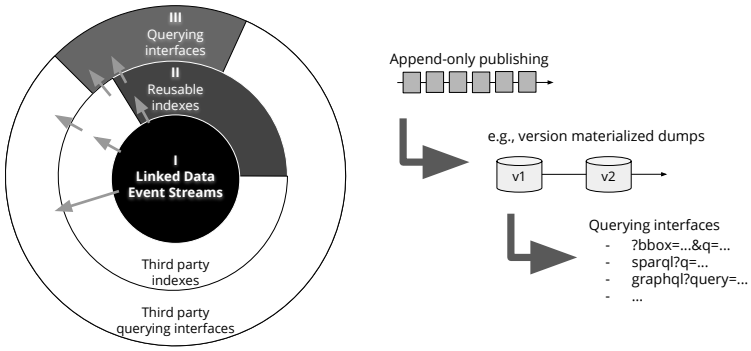

Fig. 2: The three layered shield of base registry data publishing: the core (I) is what must be done by PSBs; the second layer (II) as well as the third layer (III) can be done PSBs, but can equally be done by third parties. As tier 2 can be derived from tier 1 by anyone, and tier 3 from 2 and 1, base registry managers must first focus on the Linked Data Event Stream, then prioritize reusable indexes, and only then prioritize specific querying APIs. This will create a level playing field for an ecosystem of data services on top of this dataset.

With Linked Data Event Streams, this paper sets out a vision regarding the core task of a PSB when publishing a base registry. The LDES, which is an append-only publishing interface, is the last interface that must be removed when austerity would strike. In Fig. 2, a conceptual three layered shield illustrating the entire ecosystem sets out the next priorities. The PSB can bootstrap the ecosystem by building reusable indexes on top of their LDES by using the TREE indexer. This way, consumers – which can be both the PSB itself or third parties – can more efficiently create querying interfaces on top the dataset. When a third party for example needs an OGC API for geospatial querying, a geospatial fragmentation will allow that third party to fetch the right parts of the dataset just in time, blurring the lines between replication, prefetching and cacheable querying.

A LDES and its reusable indexes are self-descriptive thanks to the TREE Hypermedia Specification. Every page becomes part of a tree structure, and clients, such as the LDES client or Comunica, can traverse the tree to answer

certain queries. Multiple trees or indexes can be traversed in parallel, and the fastest interface for a specific task can by dynamically selected. This makes the ecosystem as a whole more resilient: there are always multiple paths to answer a certain query, in worst case having to replicate the core LDES. Contrary to the core LDES API, derived indexes can evolve faster: when a better geospatial indexes has been thought of, the old geospatial index can be taken offline without any problem.

Future work is to fully implement the TREE specification within the Comunica [5] framework to perform among others SPARQL, GraphQL-LD and autocompletion queries across Linked Data Fragments datasets. Query optimization combining interface such as TPF [8] and various TREE views and collections will be a challenge for the coming years.

References

1. Buyle, R., De Vocht, L., Van Compernolle, M., De Paepe, D., Verborgh, R., Vanlishout, Z., De Vidts, B., Mechant, P., Mannens, E.: Oslo: Open standards for linked organizations. In: Proceedings of the international conference on electronic governance and open society: Challenges in Eurasia. pp. 126–134 (2016)
2. Buyle, R., Vanlishout, Z., Coetzee, S., De Paepe, D., Van Compernolle, M., Thijs, G., Van Nuffelen, B., De Vocht, L., Mechant, P., De Vidts, B., et al.: Raising interoperability among base registries: The evolution of the linked base registry for addresses in flanders. Journal of Web Semantics **55**, 86–101 (2019)
3. Jones, J., Kuhn, W., Keßler, C., Scheider, S.: Making the web of data available via web feature services. In: Connecting a digital Europe through location and place, pp. 341–361. Springer (2014)
4. Knublauch, H., Kontokostas, D.: Shapes constraint language (shacl).(2017). W3C recommendation (2017), `https://www.w3.org/TR/shacl/#property-paths`
5. Taelman, R., Van Herwegen, J., Vander Sande, M., Verborgh, R.: Comunica: a modular sparql query engine for the web. In: International Semantic Web Conference. pp. 239–255. Springer (2018)
6. Tommasini, R., Ragab, M., Falcetta, A., Della Valle, E., Sakr, S.: A first step towards a streaming linked data life-cycle. In: International Semantic Web Conference. pp. 634–650. Springer (2020)
7. Verborgh, R., Dumontier, M.: A Web API ecosystem through feature-based reuse. Internet Computing **22**(3), 29–37 (May 2018). https://-doi.org/10.1109/MIC.2018.032501515, `https://ruben.verborgh.org/articles/web-api-ecosystem/`
8. Verborgh, R., Hartig, O., De Meester, B., Haesendonck, G., De Vocht, L., Vander Sande, M., Cyganiak, R., Colpaert, P., Mannens, E., Van de Walle, R.: Querying datasets on the web with high availability. In: International Semantic Web Conference. pp. 180–196. Springer (2014)
9. Van de Vyvere, B., Colpaert, P., Verborgh, R.: Comparing a polling and push-based approach for live open data interfaces. In: International Conference on Web Engineering. pp. 87–101. Springer (2020)
10. Wilkinson, M.D., Dumontier, M., Aalbersberg, I.J., Appleton, G., Axton, M., Baak, A., Blomberg, N., Boiten, J.W., da Silva Santos, L.B., Bourne, P.E., et al.: The fair guiding principles for scientific data management and stewardship. Scientific data **3**(1), 1–9 (2016)