Versioned Triple Pattern Fragments: 
A Low-cost Linked Data Interface Feature 
for Web Archives

Ruben Taelman, Miel Vander Sande, 
Ruben Verborgh, and Erik Mannens

Ghent University – imec – IDLab, Belgium 
(firstname.lastname)@ugent.be

Abstract. Linked Datasets typically evolve over time because triples can be removed from or added to datasets, which results in different dataset versions. While most attention is typically given to the latest dataset version, a lot of useful information is still present in previous versions and its historical evolution. In order to make this historical information queryable at Web scale, a low-cost interface is required that provides access to different dataset versions. In this paper, we add a versioning feature to the existing Triple Pattern Fragments interface for queries at, between and for versions, with an accompanying vocabulary for describing the results, metadata and hypermedia controls. This interface feature is an important step into the direction of making versioned datasets queryable on the Web, with a low publication cost and effort.

Keywords: Linked Data, versioning, Triple Pattern Fragments, Linked Data Fragments, SPARQL

1 Introduction

RDF [3] and SPARQL [8] are methods for respectively representing and querying Linked Data. The RDF data model is atemporal, as RDF itself does not define a mechanism to annotate information with dates. In practice, Linked Datasets can, however, be dynamic on different levels [10], with dataset, schema, and/or instance-level changes. These changes open up new data analysis possibilities, such as looking up data at certain points in time, requesting changes over time, or querying for times this data was valid. These querying methods are useful for example for analyzing concept drift or the management of ontology evolution.

While most data publishers currently only provide queryable access to the latest version of their datasets, data dumps of previous dataset versions are also often made available, such as the DBpedia dataset [1]. A survey on archiving Linked Open Data [16] shows that there is a need for querying over dynamic Linked Datasets at, between and for different versions.

Temporal SPARQL language extensions [7][11] at query endpoints would enable such queries, but these endpoints would at least have the same complexity as SPARQL, making it costly to make these endpoints public [21]. In order to make this version querying
possible at Web-scale, a low-cost solution such as the Triple Pattern Fragments\cite{21} (tpf) would be ideal. By itself, the tpf interface is, just like the rdf data model, atemporal. In this paper, we introduce Versioned Triple Pattern Fragments (vtpf), a Web API feature\cite{20} that provides hypermedia controls for 3 versioning queries types. For this we also introduce a new version vocabulary, which is graph-based and allows versions, changesets and versionsets to be represented.

In the next section, we introduce the requirements of the vtpf interface, after which we present related work in Section\cite{5} In Sections\cite{4} and\cite{5} we respectively introduce the version vocabulary and the vtpf interface. Finally, in Section\cite{6} the conclusions and future work are discussed.

## 2 Requirements

In order to enable queries over different dataset versions using the tpf framework, we introduced the following three requirements in previous work\cite{14}:

1. An extension of the tpf interface for versioning query types.
2. A storage solution supporting the query atoms.
3. A tpf client that is able to consume the tpf interface extension.

In this work, we introduce vtpf as a solution for the first task. We design vtpf as an interface feature rather than a separate api, in order to maximize interface compatibility and reusability\cite{20}. This means that the tpf interface remains as it is, such that regular tpf clients can continue to access it as any other tpf interface. By adding the vtpf feature, we merely provide extra possibilities for vtpf-supporting clients, which can be ignored by others. At any later stage, new or existing interface features can be added (such as other query types\cite{17} or metadata\cite{18}), while maintaining compatibility with tpf and vtpf clients. This multi-dimensional interface extensibility works because the interface explicitly describes its interaction and functionality through in-band hypermedia controls\cite{6}.

We list the following requirements for this interface, in order to make it version-aware and self-descriptive:

1. **API** support for triple pattern queries on versions, between, versions and for versions
2. **HTML** and **RDF** controls for the three versioning query types, allowing humans and machines to consume data.
3. **HTML** and **RDF** metadata to provide information about dataset versions, with links to other relevant versions.
4. **HTML** and **RDF** forms as hypermedia controls for automatic discoverability of the interface.

## 3 Related Work

Triple Pattern Fragments\cite{21} (tpf) is a Linked Data interface that provides access to Linked Datasets using triple pattern queries. By restricting access to these simple queries, the publication cost of datasets using the tpf interface is significantly reduced when
compared to SPARQL query endpoints [4]. As part of this framework, a client-side SPARQL engine has been developed that is able to consume data from TPF interfaces.

The TPF interface is based on the REST principles and makes the API browsable for machines. This is done by exposing hypermedia controls that provide declarative instructions to clients on how they can consume data from this interface, similar to how humans can understand and use visual HTML forms. TPF uses the Hydra Core Vocabulary [9] to represent these self-descriptive controls, by providing a link template explaining how triple patterns can be queried, as can be seen in Listing 1.1. Furthermore, each TPF response contains relevant metadata about the dataset and the data fragment, including an estimate of the total number of triples for that triple pattern.

In previous work [14], we explored the possibilities for adding versioning support to the TPF interface for different query atoms on storage, interface and client-level. In this paper, we focus on the following query atoms [5] related to versioning:

- **Version Materialization** (VM) queries data at a single version.
- **Delta Materialization** (DM) queries differences between two versions.
- **Version Queries** (VQ) annotate query results with versions for which they are valid.

These three query atoms correspond to realistic versioning queries, and each of them can be used to evaluate other, and more complex versioning queries [5,14].

Two TPF extensions exist that add a kind of versioning feature to the interface. The TPF Memento extension [19] supports VM queries. It does this based on the Memento protocol [12], which allows clients to select versions of HTTP resources based on content negotiation. TPF-QS [13] is an approach that supports VQ queries by annotating triples with time intervals. These two approaches are both not generic enough to support VM, DM and VQ queries. Furthermore, they both do not introduce version-oriented hypermedia controls, which is required for generic version feature discovery.

While VM queries return a regular list of triples, DM and VQ return triples with annotations. For DM queries, a method for representing deltas between two versions is required. A theoretical diff ontology [2] was explored for representing RDF graph deltas, which directly links difference graphs to other graphs. As this is only a theoretical ontology, no practically usable implementation exists. The Talis Changeset vocabulary [15] uses triple-level changesets that can contain additions and deletions, which requires reification and therefore has semantical issues. VQ queries return a combined view over all versions, and annotate each triple with a list of versions, which is related to how TPF-QS [13] compares different strategies for annotating triples with timestamps.

## 4 Vocabulary

As stated in our requirements, the VTPF interface must expose hypermedia controls and version metadata in its RDF responses. Furthermore, the three query atoms require different result structures. For these hypermedia controls, metadata and result annotation, we introduce the version vocabulary, inspired by the existing, but limited vocabularies discussed in Section 3. This vocabulary is available at [http://w3id.org/version/ontology](http://w3id.org/version/ontology), for which we use the ver: prefix.
Fig. 1: vm query html form and results.

Fig. 2: dm query html form and results.

Fig. 3: vq query html form and results.
Fig. 4 shows an overview of all available classes and properties in this new ontology, which consists of the version, changeset and versionset concepts. These last two sets are graph-based, meaning that triples can be part of a changeset or versionset graph.

The basis of this ontology is a ver:Version, which is a subclass of a void:Dataset. This means that each version of a dataset is a dataset by itself. For indicating the version of any resource, we introduce the ver:relatedVersion property. This property has several subproperties, for referencing versions before or after a version, but also for referencing the first or latest known version with relation to a dataset.

A second main class is the ver:ChangeSet, which indicates a set of added or removed triples between two versions. Each changeset must indicate the version range over which it is defined, using the ver:changeSetStart and ver:changeSetEnd properties. These properties both reference a ver:Version and are inclusive. Each changeset can either contain a list of added or a list of removed triples, where multiple changesets can exist between two versions. The ver:ChangeSetAdditions and ver:ChangeSetDeletions classes respectively indicate these types of changesets. As multiple changesets can exists, additions and deletions are always listed separately.

Finally, the ver:VersionSet class indicates a collection of triples that all exist in the same dataset versions. The versions in which the triples in this versionset exist are defined using the ver:versionSetContains property, which refers to a single ver:Version.

5 Interface

In this section, we introduce the three query atoms within the vtpf interface. We finish the section with the introduction of a live vtpf interface.

In the following subsections, we introduce new HTML and RDF forms for these query atoms, together with methods for representing their results. These three new forms are added to the existing tpf interface as an extension. Both the HTML and RDF forms have toggles for selecting the query atom, in HTML this is represented as a radio button and in RDF this is done using a versionType parameter. In order to remain backwards-compatible for clients that only support the regular tpf interface, we keep the original tpf form, but internally transform it to a vm query that selects triples against the latest dataset version.

As discussed in previous work [14], each query atom has a certain complexity when it is evaluated using a certain storage policy. Depending on this storage policy, the publisher
may want to restrict the possible query atoms that are possible through the vtpf interface. For example, a strategy with individual copies per version may be efficient for vm, but slow for dm. For the remainder of this paper, we consider a storage solution that is able to handle the three query atoms efficiently for at least triple patterns, and we therefore enable the three query atoms at the interface by-default.

5.1 Version Materialized

In order to enable triple pattern queries against specific dataset versions (vm), we added a version parameter to both the rdf and html forms.

Fig. 1 shows an example of the html form for querying triples within a certain version. This allows clients to select a version to query in, given a list of all available versions.

Listing 1.2 shows an equivalent hypermedia control for this using the Hydra Core Vocabulary [9]. This is almost equal to the existing vtpf control where only a version property was added, which can be filled in by clients. Listing 1.3 shows additional metadata that is added to the results. This metadata allows clients to determine the version of the dataset that is currently being queried, indicated using ver:relatedVersion. Furthermore, the total number of currently available versions is indicated. Finally, links to other versions are indicated, which allows clients to determine next, previous, first and last versions of datasets, and improves the discoverability of other versions.

5.2 Delta Materialized

The second main versioned query atom is for querying the differences between two versions (dm). For this, we add two parameters to the rdf and html form, for respectively selecting the start and end version.

Fig. 2 shows the html form for selecting the start and end version. Each triple is now annotated with a green “+” or a red minus “−”. “+” indicates that this triple was added somewhere inbetween the start and end versions. “−” indicates the opposite, if that triple was removed.

In Listing 1.4, the equivalent hypermedia controls are shown. Listing 1.5 shows an example of a dm query result. For dm queries, each triple is contained in a ver:ChangeSetAdditions or ver:ChangeSetDeletions graph, and respectively indicate if the triples are additions or deletions for the given version range. These two graphs are annotated with this version range in the metadata graph, as shown in Listing 1.6.

5.3 Version Query

Finally, the vq query atom selects triples over all versions, and annotates each triple with the versions in which they are present. This query atom requires no additional parameter to the rdf and html form, it only needs a flag indicating the query atom.

The html view for this query atom is shown in Fig. 3 in which all triples are annotated with their list of versions. Listing 1.7 shows an example of rdf results for vq queries. All triples are annotated with a versionset graph. These versionset graphs are annotated with the versions for which they are applicable, as can be seen in Listing 1.9.
Listing 1.1: tpf query form using the Hydra Core Vocabulary.

Listing 1.2: The vm query rdf form for triple triples in a single version.

Listing 1.3: vm query result metadata that contains links to other versions.

Listing 1.4: The dm query rdf form for triple differences over two versions.
Ruben Taelman et al.

_:changeSetDeletions {
  ol:node0
    a <http://www.w3.org/2003/01/geo/wgs84_pos#SpatialThing>;
    a <http://xmlns.com/foaf/0.1/OnlineAccount>;
    <http://xmlns.com/foaf/0.1/accountName> "mystylequicktip";
}

_:changeSetAdditions {
  ol:node0
    <http://www.w3.org/2003/01/geo/wgs84_pos#lat> "49.0000000";
    <http://www.w3.org/2003/01/geo/wgs84_pos#long> "32.0000000".
}

Listing 1.5: DM query RDF results in changesets.

http://versioned.linkeddatafragments.org/bear#metadata> {
  _:changeSetAdditions a ver:ChangeSetAdditions;
  ver:changeSetStart <http://versioned.linkeddatafragments.org/bear?
    versionType=VersionMaterialized&version=0>;
  ver:changeSetEnd <http://versioned.linkeddatafragments.org/bear?
    versionType=VersionMaterialized&version=9>.

  _:changeSetDeletions a ver:ChangeSetDeletions;
  ver:changeSetStart <http://versioned.linkeddatafragments.org/bear?
    versionType=VersionMaterialized&version=0>;
  ver:changeSetEnd <http://versioned.linkeddatafragments.org/bear?
    versionType=VersionMaterialized&version=9>.

  <http://versioned.linkeddatafragments.org/bear?versionType=VersionMaterialized
    &version=0> a ver:Version;
  ver:beforeVersion <http://versioned.linkeddatafragments.org/bear?versionType
    =VersionMaterialized&version=1>;
  ver:firstVersion <http://versioned.linkeddatafragments.org/bear?versionType
    =VersionMaterialized&version=0>;
  ver:latestVersion <http://versioned.linkeddatafragments.org/bear?versionType

  <http://versioned.linkeddatafragments.org/bear?versionType=VersionMaterialized
    &version=9> a ver:Version;
  ver:afterVersion <http://versioned.linkeddatafragments.org/bear?versionType
    =VersionMaterialized&version=8>;
  ver:firstVersion <http://versioned.linkeddatafragments.org/bear?versionType
    =VersionMaterialized&version=0>;
  ver:latestVersion <http://versioned.linkeddatafragments.org/bear?versionType
    =VersionMaterialized&version=9>
}

Listing 1.6: DM query RDF result metadata with changeset annotations.
Listing 1.7: The \texttt{vq} query \texttt{RDF} form for version-annotated triples.

\begin{verbatim}
_:versionSet0 {
  ol:node0 a <http://www.w3.org/2003/01/geo/wgs84_pos#SpatialThing>
  <http://www.w3.org/2003/01/geo/wgs84_pos#lat> "49.0000000";
  <http://www.w3.org/2003/01/geo/wgs84_pos#long> "32.0000000".
}

_:versionSet1 {
  ol:node0 a <http://xmlns.com/foaf/0.1/OnlineAccount>
  <http://xmlns.com/foaf/0.1/accountName> "mystylequicktip";
}
\end{verbatim}

Listing 1.8: \texttt{vq} query \texttt{RDF} results in versionsets.

\begin{verbatim}
_:versionSet0 a ver:VersionSet;

_:versionSet1 a ver:VersionSet;

<http://versioned.linkeddatafragments.org/bear?versionType=VersionMaterialized &version=0> a ver:Version;

\end{verbatim}

Listing 1.9: \texttt{vq} query \texttt{RDF} result metadata with versionset annotations.
5.4 Live Demo

As an example usage of the vtpf interface, we set up a live entrypoint at [http://versioned.linkeddatafragments.org/bear](http://versioned.linkeddatafragments.org/bear). This interface exposes access to the first ten versions of the rdf archive provided by the bear benchmark [5], which consists of triples compiled from the Dynamic Linked Data Observatory [1]. These first ten versions contain more than 72 million unique triples, with approximately 30 million triples per version. As backend storage solution we use ostrich [2] which supports the three query atoms for triple patterns.

6 Conclusions

In this paper, we introduced an interface for exposing query access to different versions of a dataset for triples at, between and for different different versions. This fulfills our first api requirement. For requirements 2 and 3, we provide html and rdf controls and metadata for allowing both humans and machines to understand the interface and the data. Finally, for requirement 4, we used hypermedia controls to allow machines to automatically discover how to use these query methods. As vtpf is a minimal extension to the existing tpf interface, this allows data owners to expose their different dataset versions on the Web with a low cost. This is a solution to the first task for making rdf queryable on the Web with a low publication cost.

In future work, we intend to provide solutions for the other tasks [14] for making different dataset versions possible using the tpf framework. We will extend the existing tpf client so that it is able to understand and consume data from the vtpf interface extension, for the three query atoms. For this, a variant of the sparql query language will be needed, such as t-sparql [7] or sparql-st [11]. Furthermore, our storage solution that is used as a backend to this vtpf interface will be further developed and investigated. Finally, this interface, together with the solutions for the other tasks will be evaluated, so that the publication and consumption cost with the vtpf approach can be determined.

An interface like vtpf lowers the barrier for data owners to publish different versions of their data, which will lead to the availability of more Linked Datasets at different versions over the Web. This will open up a new world of querying possibilities at, between, and for versions for domains that require such access, such as the analysis of concept drift.

---

[1] [http://swse.deri.org/dyldo/](http://swse.deri.org/dyldo)

[2] ostrich is a work-in-progress and will be introduced as a triple store with versioning support in future work.
References