Publish Data as Time consistent Web API with Provenance

Miel Vander Sande
miel.vandersande@ugent.be

Pieter Colpaert
pieter.colpaert@ugent.be

Tom De Nies
tom.denies@ugent.be

Erik Mannens
erik.mannens@ugent.be

Rik Van de Walle
rik.vandewalle@ugent.be

Ghent University - iMinds
Department of Electronics and Information Systems, Multimedia Lab
Gaston Crommenlaan 8 bus 201
B-9050 Ledeberg-Ghent, Belgium

ABSTRACT

Many organisations publish their data through a Web API. This stimulates use by Web applications, enabling reuse and enrichments. Recently, resource-oriented APIs are increasing in popularity because of their scalability. However, for organisations subject to data archiving, creating such an API raises certain issues. Often, datasets are stored in different files and different formats. Therefore, tracking revisions is a challenging task and the API has to be custom built. Moreover, standard APIs only provide access to the current state of a resource. This creates time-based inconsistencies when they are combined. In this paper, we introduce an end-to-end solution for publishing a dataset as a time-based versioned REST API, with minimal input of the publisher. Furthermore, it publishes the provenance of each created resource. We propose a technology stack composed of prior work, which versions datasets, generates provenance, creates an API and adds Memento Datetime negotiation.

1. INTRODUCTION

Publishing data on the Web is considered a fruitful activity in many domains, driven several motivations. On the one hand, they publish data because they are obliged to, as acknowledgement to their public funding. On the other hand, the dissimination of their data enables them to extract knowledge from reuse by third-parties. For instance, the cultural heritage community has been sharing metadata over the Web for years as they maintain long term archives. Furthermore, the governments, pushed by the growing Open Government Data initiative, make their data publicly available, too.

In order to achieve this, the use of Web applications is required, which, in their turn, require machine consumable data access. Therefore, installing a Web API is usually considered best practice and trivial task. Furthermore, these characteristics affect the API implementation as well. In principle, a Web API can be built according to different architectural styles (e.g., RPC or REST). However, in the past years, REST APIs have increased in popularity in spite of others [6]. The main basis is scalability and interoperability between systems. Its resource-oriented architecture is helpful in mashups (e.g., Yahoo Pipes), where data from different APIs can automatically be composed [12].

Nevertheless, as Web APIs are implemented at the moment, they expose only the latest version of the available data. In public organisations, it is common to store snapshots of datasets over a long period of time, together with time-based metadata. Furthermore, in most cases, the data is usually stored in several files, using different formats. All these different formats should be equally exposed to the end users. Moreover, these datasets are curated collaboratively by different people. Therefore, it is needless to say, that this requires a good data management solution supporting file-based *versioning*. Those versions, present in the data, are not propagated: the API access is usually limited to the latest version.

When composing services, a static state over resources is desirable. As they are implemented now, resources change state constantly and can cause inconsistencies when used out of sync. For example, if one combines a resource "Unemployment rate in the US", last updated in 2004, with the resource "Average current income", last updated in 2012, the combined information would be less reliable than data collected for the same year. Therefore, publishers and their consumers can also benefit from exposing prior versions as resources as well. This is mainly because of the characteristics of the underlying data management. Furthermore, because of the data's heterogeneity, such an API has to be custom built.

In this paper, we describe an end-to-end solution for creating a Web API exposing any dataset, while providing versioning in the time dimension over HTTP. We propose a complete technology stack composed of previous work, which covers each requirement mentioned above. We start by using a version control system to track file changes and generate provenance from its metadata. Then, we use a data publishing tool to automatically create an API from the provenance descriptions. Finally, we extend the API with time-based versioning support for its resources by adding a special proxy server. It exposes prior states of a resource by using hypermedia, allowing the client to do Datetime negotiation. As a result, data can be published in a small amount of time.

This paper is structured as follows. First, we introduce the different components in our stack in Section 2 and discuss related work in Section 5. Next, Section 3 describes how we can set up a time-based versioned Web API from data files. Then, we demonstrate our approach with an example in Section 4. We end by discussing future work in Section 6 and close with conclusions in Section 7.

WWW'14 Companion,, April 7–11, 2014, Seoul, Korea.
ACM 978-1-4503-2745-9/14/04.
http://dx.doi.org/10.1145/2567948.2579217.

2. TECHNOLOGY STACK

The work described in this paper combines three tools: *Git2PROV* [4], *The DataTank* [16], and *Generic Memento* [18]. In this section, we introduce each of these tools briefly.

2.1 Git2PROV

Version Control Systems (VCS), such as *git*[1], are used to facilitate collaboration for several types of content, primarily code and data. They generally consist of repositories, in which the data and all its history is stored. Because of this, these VCS repositories contain an abundance of provenance information. *Git2PROV*[2] is a service that maps this information inside *git* repositories to standardized provenance information, following the standard W3C PROV Data Model [19]. The information mapped by *Git2PROV*, includes creation and deletion times, as well as dependencies and derivations of versioned files.

2.2 The DataTank

The DataTank[3] is a project which provides a HTTP API out of various data sources. It does not store data: each time a resource is requested, the data is fetched in the memory on the server and a transformation is done towards an accepted content-type. It can be used by organizations to publish datasets on The Web which are stored in, amongst others, excel files, CSV files, triple stores and shape files. The API of The DataTank allows user agents to add dataset configurations, to add metadata fields, to export a metadata feed described in DCAT [4] [9], etc. Furthermore, the source readers can be extended with custom code to read any kind of source, or combine different sources into one resource.

2.3 Generic Memento

As mentioned in the introduction, the resulting API relies on the *Memento* framework [13] to provide time-based versioning over HTTP. In this stack, we use our prior work *Generic Memento*. This works allows implementing Memento functionality as a generic proxy server, based on published provenance. In this section, we describe both frameworks in more detail and clearify the distinction between the two.

2.3.1 The Memento framework

The Memento framework aims at a tighter integration between the current and the past Web. It introduces an extra dimension for *content negotiation* between client and server: the *datetime* dimension. The framework defines a set of resource types, their characteristics and the relation between them. A resource, of which prior states are desired, provides the current state and is referred to as the *Original Resource URI-R*. For *URI-R*, prior states at time t_i – if they exist – are encapsulated in distinct resources referred to as *Mementos URI-*$M_i(i = 1..n)$. This distinction allows *URI-R* to keep a stable URI, since only old versions get a new one. To allow caching *URI-R*, it is disconnected from *URI-*M_i. Therefore, *URI-R* cannot be datetime negotiated directly, but is done on a third resource *URI-G*, called the *TimeGate*. All three resources are connected using hypermedia, as shown in Figure 1, which enables navigating between them. We demonstrate this with a typical client-server interaction below.

[1] `http://git-scm.com/`
[2] `http://git2prov.org`
[3] `http://thedatatank.com`
[4] A vocabulary standardized by the W3C to describe data catalogs.

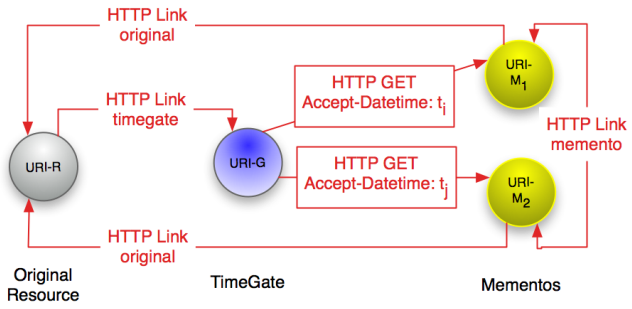

Figure 1: The memento framework consist of three resources: Original Resource *URI-R*, the TimeGate *URI-G* and the Mementos *URI-*M_i. They are linked using hypermedia. Source: `http://www.mementoweb.org/`

1. When a client requests *URI-R*, the servers response holds a HTTP Link header of type `timegate` pointing at *URI-G*, which enables its discoverability.
2. The client can use *URI-G* to do *datetime* negotiation. The request is sent holding a `Accept-Datetime:` t_r header, referring to the desired state valid at that time. Then, the server decides on the best matching Memento *URI-*M_x with creation time t_x, with $t_x = max(\forall\ t_i <= t_r)$. The selected *URI-*$M_x$ is returned in the HTTP Location header and with response code `302 Found`, which redirects the client.
3. By following the redirect to *URI-*M_x, the server returns the prior state of *URI-R*. The response contains a HTTP Link header of type `original` pointing back at *URI-R* for discoverability. Additionally, links to the `first-memento`, `next-memento` and `prev-memento` relationships can be added to the headers as well.

A more in-depth description including detailed examples can be found in the Memento Guide[5].

Unfortunately, Memento requires a custom implementation for each server. APIs are written in different languages, serve different resources and work on a different storage layer. The next section describes how this drawback can be avoided in creating a generic memento proxy server.

2.3.2 A Generic Memento proxy

A *Generic Memento* proxy server enables Memento functionality to Linked Data servers, or any other *resource-oriented* API, that publishes provenance about their resources. Provenance can formally describe how one prior state relates to the next one and when the change occurred. The proxy's main contribution is creating a loose coupling between the API and its Memento functionality. Based on the provenance descriptions, it decides on matching *Mementos* and forwards the requests accordingly. This lowers implementation costs and increases reuse.The decision logic is implemented as a set of N3 [2] rules, executed on the provenance by the semantic reasoner EYE [5].

The general architecture is shown in Figure 2 and consists of two independent types of services: a *Linked Data Service* (LDS) and a *Generic Memento Proxy* (GMP). The former publishes a Linked Data resource *URI-L* and their provenance in PROV *URI-P*, which describes the archives *URI-*A_i. The latter provides all the original functionality of the Memento framework, including access to the resources *URI-R*, *URI-*M_i and *URI-G*. We retake the example from the previous section to demonstrate the functionality and distinction.

[5] `http://www.mementoweb.org/guide/`

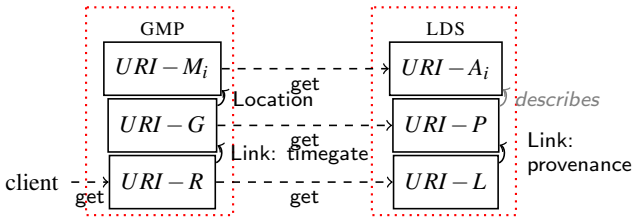

Figure 2: The Generic Memento proxy GMP and the Linked Data service LDS are loosely coupled, creating a generic architecture.

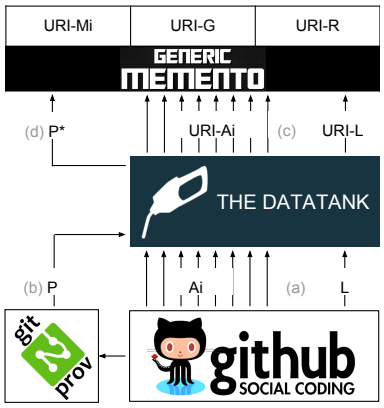

Figure 3: The source file L and its prior versions A_i are accessible through *GitHub*, while *Git2PROV* generates provenance in PROV from the *git* metadata. Based P, *The DataTank* publishes L and A_i as API, exposes them as resources *URI-L* and *URI-*A_i, and described their relation in P as P^*. Finally, *Generic Memento* uses P^* to create the Memento resources *URI-R*, *URI-*M_i, *URI-G*

1. When a client requests *URI-R*, the proxy will forward the request to *URI-A*. it will add the *timegate* Link header to the response and send it back to the client.
2. When the client does *datetime* negotiation on *URI-G*, the proxy will request *URI-P*. Possible discovery for *URI-P* is through a `provenance` Link header in the response of *URI-L* [10]. The reasoning is executed and the matching resource *URI-*A_x is retrieved from the provenance chain. Each resource *URI-*A_i is encapsulated in a corresponding resource *URI-*M_i, so the necessary headers can be added, as mentioned before. The selected *URI-*M_x is returned in the HTTP Location header and with response code `302 Found`, which redirects the client.
3. When the client follows the redirect to *URI-*M_x, the proxy forwards the request towards its corresponding *URI-*A_x. Its response is extended with the `original` Link header and optional `first-memento`, `next-memento` and `prev-memento` relationships.

For more in-depth information about *Generic Memento*, we refer to the original paper [18].

3. OPERATIONAL STEPS

In this section, we describe in four steps how the proposed technology stack (as described in Section 2) can publish data as a Web API, supporting Datetime negotiation. The layout of the stack and the interaction between components are shown in Figure 3.

3.1 Versioning Data

Version control systems are extremely helpful in managing, revising and collaboratively writing files of source code. Amongst such tools, *git* has gained a tremendous amount of popularity. Git is a fast, scalable, distributed revision control system with a rich command set that provides both high-level operations and full access to internals [15]. In this approach, we use *GitHub*[6], a publicly hosted *git* service with an extra Web API. This creates a dereferencable URI for each file version in the repository. We will call a file stored in a *git* repository L and its versions A_i (Figure 3a).

Although its limitations for big datasets, *git*, in combination with *GitHub*, has already been used as the engine behind a couple of Open Data Ecosystems [11] . For instance, the city of Montpellier[7], the city of Chicago[8] and Open Knowledge Foundation (OKFN) [9] created *GitHub* accounts to publish data in various formats. The metadata are stored in README files: who is the owner, when was it published, category, description, license, and so on. Examples of publishing the lawbook using *git* exist in Germany[10], The Netherlands[11] and Flanders[12]. Publishing Linked Data through *git* has been done using serialized RDF. For Instance, Ross Singer published the description of MARC codes, a US standard for bibliographic catalogues, as separate Turtle files[13].*GitHub* stimulates using its service for datasets by supporting extra features for GeoJSON and CSV[14].

3.2 Generating Provenance

The *Git2PROV* tool takes a git repository URI as input, and outputs its metadata as valid PROV, which we refer to as P (Figure 3b). This provenance describes the relations between L and A_i. A partial output is shown in Listing 1. *Git2PROV* expresses provenance in PROV-DM concepts by identifying three classes in the commit:

Dependency The dependency between two files is modelled as `prov:wasDerivedFrom` between two `prov:Entity` objects.

Activity The commit action is expressed as `prov:Activity`, connecting two `prov:Entity` objects through the relations `prov:used` and `prov:wasGeneratedBy` relations.

Attribution The author and committer are both a `prov:Agent`, linked to the created `prov:Entity` using `prov:AttributedTo` for the former and `prov:wasAssociatedWith` for the latter.

The next section describes how P is used together with *The DataTank* to create the dereferencable URIs *URI-L* and *URI-*A_i for L and A_i.

3.3 Creating the API

Once our repository is ready and the provenance is generated, we create a data API using *The DataTank*. For L and A_i, it creates the corresponding resources *URI-L* and *URI-*A_i, as shown in Figure 3c. *The DataTank* creates a data adapter between the original data source and the data consumer (e.g., App developer). According to the hierarchy present in the data, a set of resources is created allowing the data to be traversed in a directory-style manner. This makes the original set fully or partially accessible.

Sources are added using *The DataTank*'s own Web API by creating a so called *resource definition*. Such a definition is a collection

[6] `https://github.com/`
[7] `https://github.com/VilleDeMontpellier/`
[8] `https://github.com/Chicago/`
[9] `https://github.com/datasets`
[10] `https://github.com/bundestag/gesetze`
[11] `https://github.com/statengeneraal/wetten-tools`
[12] `https://github.com/okfn-be/codex-vlaanderen`
[13] `https://github.com/rsinger/LinkedMARCCodes/`
[14] `http://government.github.com/stories/forking-your-city/`

```
1  @prefix prov:  <http://www.w3.org/ns/prov#>.
2  @prefix xsd:  <http://www.w3.org/2001/XMLSchema#>.
3  @prefix rdfs:  <http://www.w3.org/2000/01/rdf-schema#> .
4  @prefix result:  <http://git2prov.org/git2prov?giturl=https%3A%2F%2Fgithub.com%2Fdatasets%2Fcountry-codes.git&serialization=PROV-O#>
   .
5  result:commit-1c036643ef668ef836f251ead1cdd0835dbfdb3b a prov:Activity ;
6    prov:endedAtTime               "2013-12-09T09:03:46.000Z"^^xsd:dateTime ;
7    prov:wasAssociatedWith         result:user-ewheeler .
8
9  result:commit-ff1406b10dd4f484376db0c27b9c5d92643ef22e a prov:Activity ;
10   prov:endedAtTime               "2013-12-09T10:02:48.000Z"^^xsd:dateTime ;
11   prov:wasAssociatedWith         result:user-ewheeler .
12
13 result:file-data-country-codes-csv       a prov:Entity ;
14   rdfs:label    "data/country-codes.csv"@en .
15
16 result:file-data-country-codes-csv_commit-1c036643ef668ef836f251ead1cdd0835dbfdb3b a prov:Entity ;
17   prov:specializationOf result:file-data-country-codes-csv ;
18   prov:wasAttributedTo  result:user-ewheeler ;
19   prov:wasGeneratedBy   result:commit-1c036643ef668ef836f251ead1cdd0835dbfdb3b .
20
21 result:file-data-country-codes-csv_commit-ff1406b10dd4f484376db0c27b9c5d92643ef22e a prov:Entity ;
22   prov:specializationOf result:file-data-country-codes-csv ;
23   prov:wasAttributedTo  result:user-ewheeler ;
24   prov:wasDerivedFrom   result:file-data-country-codes-csv_commit-1c036643ef668ef836f251ead1cdd0835dbfdb3b ;
25   prov:wasGeneratedBy   result:commit-ff1406b10dd4f484376db0c27b9c5d92643ef22e .
```

Listing 1: A partial example of the *Git2PROV* output for the OKFN Country Codes dataset[9]: the metadata stored in *git* commits are transformed into PROV descriptions

of metadata, i.e., location of the source, the format, format-specific parameters (e.g., delimiter). The definition is serialised in JSON and added to send in the body of a PUT request. This request is sent to the *definitions collection* (`http://{datatank}/api/definitions/`), appended with an identifier. Its Content-Type header contains the media type `application/tdt.definition+json`. The stored definition creates an API, whose base URI is returned in the Location header. For instance, if L is a CSV file with delimiter ',', *URI-L* is created with the following request:

```
PUT /api/definitions/{identifier} HTTP/1.1
Content-Type: application/tdt.definition+json

{"type": "text/csv", "url": "L", "delimiter": ","}
-------------------------------------------------
HTTP/1.1 200 OK
Location: http://{datatank}/{identifier}
```

From the provenance P, generated by *Git2PROV*, we can assemble resource definitions to create *URI-L* and *URI-*A_i, based on L and A_i. These dereferencable URIs are required by *Generic Memento*, as discussed in Section 3.4. In P, the datasets from the repository are included with a specific URI and are typed as `prov:Entity`. This URI contains the file name and the commit hash, which we use to reconstruct the *GitHub* URL giving direct access to a certain version. For each extracted *GitHub* URL L or A_i, we create a source definition and add it to our *The DataTank* instance. Each created URI is added to the provenance with an `owl:sameAs` link to its corresponding file, resulting in extended provenance P^*.

3.4 Adding datetime negotiation

Finally, we will add Memento support to our created resources by feeding P^* to a *Generic Memento* proxy server (Figure 3d). We decide on a Memento *URI-*M_i using semantic reasoning. The resulting rules are demonstrated in Listing 2. First, we identify all Mementos. Each revision is linked to its predecessor using the predicate `prov:wasRevisionOf`, forming a chain of revisions with *URI-R* as endpoint. Relying on the transitive property defined in OWL logic, the relation between *URI-R* and *URI-*M_i is derived by adding the triples on lines 5 and 6.

Next, we select a version valid at a given datetime, which is added by the triple at line 9. The predicate `prov:wasGeneratedBy` links each version to an instance of `prov:Activity`, whose `prov:endedAtTime` predicate indicates the initiation of validity. The rule starts with extracting the defined datetime *[line 12]* and creating a finite list of occuring datetimes. This list is composed by finding all datetimes *[line 13]* that occur on or before the requested datetime *[line 16]*. The valid version occurs on the latest datetime in that list *[lines 18 and 19]*, and is added to the response *[line 21]*. In addition, we define analogue rules to select the first, the last, the next and the previous Memento as well, since links to all of these resources are required. The complete rule file can be found here: `http://goo.gl/dz13UN`.

After the rule execution, the derived result can be mapped directly to the response. We add a *Location* header pointing at M_i. We add a Link header, if applicable, to the first Memento *URI-*M_0, the next Memento *URI-*M_{i+1}, the previous Memento *URI-*M_{i-1}, as well as the Original Resource *URI-R*.

4. EXAMPLE: OKFN COUNTRY CODES

In this section, we demonstrate our approach with a detailed example, applying each step defined in Section 3 to a real-world example.

1. Versioning Data.

As sample dataset, we use the Country Codes CSV file in the OKFN Country Codes repository[15] L_{CC}. At the time of writing, the file was subject to two commits:

- Commit C_x with hash $x =$ *1c036643ef668ef836f251ead1c...db3b* created at $t_x =$ *2013-12-09T09:03:46.000Z*, which creates a version A_{CCx}

[15] `https://github.com/datasets/country-codes`

```
@prefix prov:  <http://www.w3.org/ns/prov#>.
@prefix pred:  <http://www.w3.org/2007/rif-builtin-predicate#>.
@prefix xsd:  <http://www.w3.org/2001/XMLSchema#>.
@prefix e:  <http://eulersharp.sourceforge.net/2003/03swap/log-rules#>.
prov:wasRevisionOf rdfs:subPropertyOf :memento.
:memento a owl:TransitiveProperty.

:request :datetime
        "2012-04-11T12:30:00Z"^^xsd:dateTime.

{
  :request :datetime ?req_datetime.
  [] e:findall (?datetime {
    ?rev prov:endedAtTime ?datetime .
    (?datetime ?req_datetime)
       pred:dateTime-less-than-or-equal true.
  } ?datetime_list) .
  ?datetime_list e:max ?current_datetime.
  ?current prov:endedAtTime ?current_datetime.
} => {
  :response :memento ?current.
}.
...
```

Listing 2: N3Logic selects the Memento valid at datetime 2012-04-11T12:30:00Z

- Commit C_y with hash $y = ff1406b10dd4f484376db0c27b9...f22e$ created at $t_y = 2013\text{-}12\text{-}09T10:02:48.000Z$, which creates a version A_{CCy}

2. Generating Provenance.

We generate its provenance P_{CC} by adding the repository using *Git2PROV*[16]. An extract of its output is shown in Listing 1.

3. Creating the API.

Next, we set up a *The DataTank* instance on a local server (`localhost`), for which we defined the following resources:

- Resource Definition *URI-RD*$_{CC}$: `http://localhost/api/definitions/okfn/country-codes`
- Resource Definition *URI-RD*$_{CCx}$: `http://localhost/api/definitions/okfn/country-codes-x`
- Resource Definition *URI-RD*$_{CCy}$: `http://localhost/api/definitions/okfn/country-codes-y`
- Resource *URI-L*$_{CC}$ publishing L_{CC}: `http://localhost/okfn/country-codes`
- Resource *URI-A*$_{CCx}$ publishing A_{CCx}: `http://localhost/okfn/country-codes-x`
- Resource *URI-A*$_{CCy}$ publishing A_{CCy}: `http://localhost/okfn/country-codes-y`

As shown in Listing 3, adding *URI-RD*$_{CC}$, *URI-RD*$_{CCx}$ and *URI-RD*$_{CCy}$ to our *The DataTank* instance creates the resources *URI-L*$_{CC}$, *URI-A*$_{CCx}$ and *URI-A*$_{CCy}$, returned in the HTTP Location header of each reponse. Next, we add `owl:sameAs` links to P_{CC} between each `prov:Entity` and its corresponding resource. For instance, *URI-L*$_{CC}$ is added with the following triple:

```
result:file-data-country-codes-csv
owl:sameAs
<http://localhost/okfn/country-codes>
```

[16] `http://git2prov.org/git2prov?giturl=https%3A%2F%2Fgithub.com%2Fdatasets%2Fcountry-codes.git&serialization=PROV-O`

```
1: UA ——— HTTP PUT; Content-Type:
          application/tdt.definition+json
          {"type": "text/csv",
           "uri": "L_CC", "delimiter": ","} ——> URI-RD_CC
2: UA <— HTTP 200; Location: URI-L_CC ———— URI-RD_CC

3: UA ——— HTTP PUT; Content-Type:
          application/tdt.definition+json
          {"type": "text/csv",
           "uri": "A_CCx", "delimiter": ","} ——> URI-RD_CCx
4: UA <— HTTP 200; Location: URI-A_CCy ———— URI-RD_CCx

5: UA ——— HTTP PUT; Content-Type:
          application/tdt.definition+json
          {"type":"text/csv",
           "uri": "A_CCy","delimiter":","} ——> URI-RD_CCy
6: UA <— HTTP 200; Location: URI-A_CCy ———— URI-RD_CCy
```

Listing 3: Resource definitions are added to *The DataTank* with sending PUT request to the API, creating the required resources

```
1:  UA ——— HTTP GET/HEAD; Accept-Datetime: t_j —> URI-R_CC
2:  URI-R_CC ———— HTTP GET/HEAD ————> URI-L_CC
3:  URI-R_CC <———— HTTP 200 ———————— URI-L_CC
4:  UA <——— HTTP 200; Link: URI-G_CC ———— URI-R_CC

5:  UA ——— HTTP GET/HEAD; Accept-Datetime: t_j —> URI-G_CC
6:  UA <—— HTTP 302; Location: URI-M_x; Vary;
                Link: URI-R_CC, URI-G_CC, URI-M_CCy —— URI-G_CC

7:  UA ——— HTTP GET; Accept-Datetime: t_j ————> URI-M_x
8:  URI-M_x ———— HTTP GET ————————> URI-A_x
9:  URI-M_x <———— HTTP 200 ———————— URI-A_x
10: UA <—— HTTP 200; Memento-Datetime: t_j;
                Link: URI-R_CC, URI-G_CC, URI-M_CCy —— URI-M_x
```

Listing 4: A valid state of our dataset at time t_j = 2013-12-09T09:30:00.000Z can now be accessed through an API using only HTTP interaction

4. Adding Datetime negotiation.

As a final step, we have a *Generic Memento* proxy running at the location `http://localhost/memento/`. We register the resource *URI-L*$_{CC}$ and supply its provenance P^*_{CC}. The following resources are created as a result:

- *Original Resource URI-R*$_{CC}$: `http://localhost/memento/okfn/country-codes`
- *Timegate* resource *URI-G*$_{CC}$: `http://localhost/memento/tmgate/okfn/country-codes`
- *Memento* resource *URI-M*$_{CCx}$, proxying *URI-A*$_{CCx}$: `http://localhost/memento/okfn/country-codes/x`
- *Memento* resource *URI-M*$_{CCy}$, proxying *URI-A*$_{CCy}$: `http://localhost/memento/okfn/country-codes/y`

The HTTP interaction between a User Agent (UA) requesting the valid state of resource *URI-R*$_{CC}$ at time $t_j = 2013\text{-}12\text{-}09T09:30:00.000Z$ is demonstrated in Listing 4. First, the agent requests the resource from the proxy *[Line 2]*, which forwards the request to the corresponding resource *URI-L*$_{CC}$ from *The DataTank* instance *[Line 3]*. When the response returns, the proxy adds a Link header pointing to *URI-G*$_{CC}$ *[Line 3,4]* to provide the Datetime negotiation functionality. Next, the agent sends a request to *URI-G*$_{CC}$ adding an Accept-DateTime header holding the desired timestamp t_j *[Line 5]*. The memento *URI-M*$_{CCx}$ is selected by the proxy, since t_x is the highest timestamp occurring before t_j, and sent back in the Location header *[Line 6]*. Finally, the agent requests the memento *URI-M*$_{CCx}$ *[Line 7]*, which forwards the request to *URI-A*$_{CCx}$ *[Line 8]*. The proxy adds extra Link headers to the response and sends it back to the agent *[Line 9,10]*.

5. RELATED WORK

The closest related work is Camlistore[17], an archiving storage space. Users can add and modify *blobs* of data, of which changes are stored over time. The stored data is accesible through several interfaces, including a Web API. Although it serves data over HTTP, it does not allow fine-grained access to the data, nor does it offer resource-based versioning. Furthermore, it is an integrated, centralized solution. The components of the stack described in this paper can work distributed, i.e. not every party needs to offer the complete stack. Also, it does not publish provenance, which enables obtaining trust.

Also related are server-side implementations of the Memento framework. The Wayback software [14] is a webpage archiving system that stores snapshots of a registered resource's representation, and assignes a new URI to it. Since this system relies on snapshots, it does not contain all the versions of a resource. The SiteStory extension for Apache [1] webservers works in a similar way, but allows more fine-grained archiving. A snapshot is made each time the data is requested, approximating a full history. However, only the last occuring change between two requests is stored, loosing the others. These systems rely on *polling*, i.e. snapshots are made on certain points in time, while our approach is *data-driven*: every change to the data is reflected in the Memento interface. Another data-driven implementation is the Memento MediaWiki extension [8]. The history of content changes is automatically exposed as Memento resources. However, this solution is a MediaWiki-specific implementation.

In the context of provenance, an extension to the Memento framework was proposed to publish provenance [3]. For each published resource by the framework, a Link header of type *provenance* is added to its response for discovery. This work is complementary to the *Generic Memento* proxy, valuable for republishing the provenance for each *Memento*.

6. FUTURE WORK

In the future, we will research the applicability in the Linked Data publishing domain. In prior work, we have developed R&Wbase[17], a distributed version controlled triple store. Instead of storing snapshots of the whole triple set, only deltas are stored according to a numbering scheme. This scheme enables a fast and lightweight retrieval algorithm, that allows real-time version retrieval. PROV is used at operation level to describe the relation between the different versions. In other words, the *git* storage layer described in this paper would be replaced by a R&Wbase instance. Additionally, we will study the capabilities of Memento as a versioning access method for known triple store interfaces like SPARQL [7].

7. CONCLUSION

In this paper we introduced a fast and flexible method for publishing datasets as a time-based versioned Web API. It facilitates publishing data on the Web, which is an advantageous activity in many domains, e.g., Cultural Heritage and Open Government Data. The potential lies in reusing and combining these data to extract information. Our work addresses three main issues for such organisations: simplifying the creation of a *resource-oriented* RESTAPI, supporting constant state over different combined resources, and disseminating provenance. All components of the stack, and their connections, were discussed in detail.

[17]http://camlistore.org/

8. REFERENCES

[1] Balakireva, L.: The SiteStory extension for Apache (2013), http://mementoweb.github.io/SiteStory

[2] Berners-Lee, T., Connolly, D.: Notation3 (N3): A readable RDF syntax. W3C Team Submission (Mar 2011), available at http://www.w3.org/TeamSubmission/n3/

[3] Coppens, S., Mannens, E., Van Deursen, D., Hochstenbach, P., Janssens, B., Van de Walle, R.: Publishing provenance information on the Web using the Memento datetime content negotiation. In: WWW2011 workshop on Linked Data on the Web (LDOW 2011). vol. 813, pp. 6–15 (2011)

[4] De Nies, T., Magliacane, S., Verborgh, R., Coppens, S., Groth, P., Mannens, E., Van de Walle, R.: Git2prov: Exposing Version Control System Content as W3C PROV. In: Poster and Demo Proceedings of the 12th International Semantic Web Conference (2013)

[5] De Roo, J.: Euler Proof Mechanism (1999–2013), available at http://eulersharp.sourceforge.net/

[6] DuVander, A.: 3,000 web APIs: Trends From a Quickly Growing Directory (2011), http://blog.programmableweb.com/2011/03/08/3000-web-apis/

[7] Garlik, S.H., Seaborne, A., Prud'hommeaux, E.: SPARQL 1.1 Query Language. World Wide Web Consortium (2013), http://www.w3.org/TR/sparql11-query/

[8] Jones, S., Shankar, H.: The Memento extension for MediaWiki (2013), http://www.mediawiki.org/wiki/Extension:Memento

[9] Maali, F., Erickson, J., Archer, P.: Data Catalog Vocabulary (DCAT). W3C Working Draft (2012)

[10] Moreau, L., Hartig, O., Simmhan, Y., Myers, J., Lebo, T., Belhajjame, K., Miles, S.: PROV-AQ: Provenance Access and Query. Tech. rep. (2012), http://www.w3.org/TR/prov-aq/

[11] Polock, R.: Building the Open Data Ecosystem (2011), http://blog.okfn.org/2011/03/31/building-the-open-data-ecosystem/

[12] Rao, J., Su, X.: A survey of automated Web Service composition methods. In: Semantic Web Services and Web Process Composition, pp. 43–54. Springer (2005)

[13] Van de Sompel, H., Sanderson, R., Nelson, M.L., Balakireva, L., Shankar, H., Ainsworth, S.: An HTTP-based Versioning Mechanism for Linked Data. CoRR abs/1003.3661 (2010)

[14] Tofel, B.: Wayback (2011), http://archive-access.sourceforge.net/projects/wayback

[15] Torvalds, L.: git(1) Manual Page (2013), https://www.kernel.org/pub/software/scm/git/docs/

[16] Vander Sande, M., Colpaert, P., Van Deursen, D., Mannens, E., Van de Walle, R.: The DataTank: an Open Data adapter with Semantic output. In: 21st International Conference on World Wide Web, Proceedings. p. 4 (2012)

[17] Vander Sande, M., Colpaert, P., Verborgh, R., Coppens, S., Mannens, E., Van de Walle, R.: R&Wbase: git for triples. In: Proceedings of the 6th Workshop on Linked Data on the Web (May 2013)

[18] Vander Sande, M., Coppens, S., Verborgh, R., Mannens, E., Van de Walle, R.: Adding Time to Linked Data: A Generic Memento proxy through PROV. In: Poster and Demo Proceedings of the 12th International Semantic Web Conference (2013)

[19] W3C Provenance Working Group and others: PROV-DM: The PROVData Model (L. Moreau & P. Missier, Eds.). (2012)