DRPD: Decentralizing RDF prefixes

Dominik Tomaszuk and Karol Litman

Institute of Informatics, University of Bialystok
ul. Ciolkowskiego 1M, 15-245 Bialystok, Poland
d.tomaszuk@uwb.edu.pl

Abstract. Internationalized Resource Identifier (IRI) references are an inseparable part of the Semantic Web. Sometimes the references are difficult to remember, so many Resource Description Framework (RDF) serializations, e.g. Turtle, allow to shorten them. This paper describes how to find the appropriate prefix, and resolved them to the full IRI reference. Many endpoints that provide information about prefixes and namespaces can exist in our architecture. The tools that we present are not as sensitive to failures as centralized solutions and let users use multiple prefix endpoints at the same time.

Keywords: RDF prefix · namespaces · decentralization · Turtle · RESTful web service · API · intelligent clients

1 Introduction

Since Internationalized Resource Identifier (IRI) references can be long, prefixes are used to create a mapping between the IRI and a namespace prefix. This mapping enables the abbreviation of IRIs, therefore it achieves a more convenient way to read and write Resource Description Framework (RDF) documents.

Prefixes occur in various forms in different RDF serializations. Probably the most popular type of prefix occurs in Turtle [8] and it is called *prefixed names*. A prefixed name consists of a prefix label and a local part, separated by a `:` sign. The `@prefix` directive[1] associates a prefix label with an IRI. RDF/XML [9] supports *qualified names* [3] (so-called QNames), which is subset of Turtle's prefixed names. To shorten IRIs, RDFa [10] uses *CURIEs* [1] (so-called Compact URIs). Unlike QNames, the part of a CURIE after the `:` sign does not need to conform to the rules for XML element names. A similar mechanism is also found in JSON-LD [7] and it is called a *context*. The cotext is used to map short words (so-called terms) to IRIs.

RDF developers often use IRI references, so they need a tool to remember and look up prefixes. To provide meaningful prefixes for unknown vocabularies the prefix.cc web page[2] can be used. Unfortunately, this application is centralized. In this paper we present DRPD tools that are decentralized. Moreover, we propose a serialization and an architecture for decentralization.

[1] In Turtle 1.1 we can use SPARQL style `PREFIX` directive as well.
[2] http://prefix.cc/

The paper is constructed according to sections. In Section 2 we present our architecture and serialization. In Section 3 we demonstrate our tools and discuss their user interface. The paper ends with conclusions.

2 System Architecture

The system architecture consists of a database (see Subsection 2.1), web service (see Subsection 2.2), console application (Subsection 2.3) and web application (Subsection 2.4). All elements of the system form a coherent whole and are based on the RESTful web service. The system architecute is presented in Fig. 1. The main element of architecture is web service with open API. Different clients (e.g. console application, web application) can use any endpoints that is compatible with the API described below. In Subsection 2.3 we also propose Turtle$_d$, a serialization that extends Turtle to support many endpoints resolving prefixes.

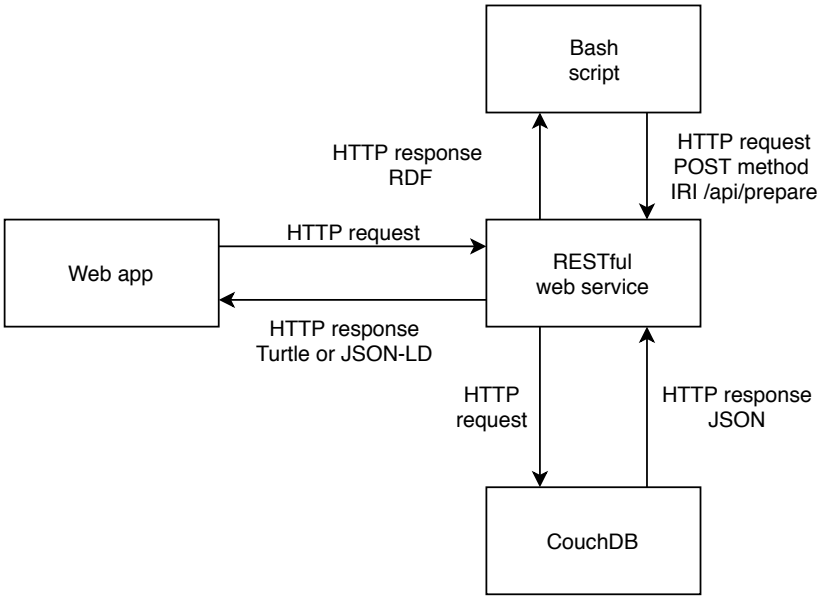

Fig. 1. DRPD architecture

2.1 Database

The first part that collects data is a database based on JSON documents and on the REST architecture [4] which is CouchDB [2]. Each occurring document in the database is an object representing a namespace. The objects have such fields as: `namespace`, `prefix`, `pluses`, `minuses`, and `score`. The namespace is a

unique field that identifies a JSON document, and a prefix can be joint to many documents. If there are two documents containing the same prefix, it means that there are two different namespaces. The pluses field and the minuses field have a complex structure defined by arrays that separate the logins of signed in users into individual authentication servers. Storing it in one object could cause collision of identifiers from different authentication servers. The score field is separate for each namespace due to the web service can set the namespaces in a given prefix in a chronological order.

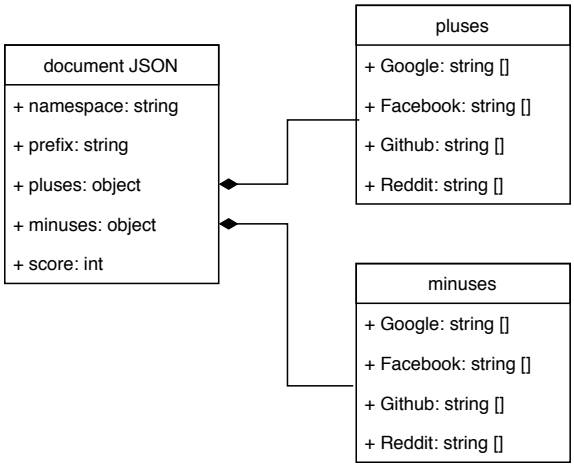

Fig. 2. Database diagram

2.2 RESTful web service

The main part of our architecture is the web service, which is based on REST architecture [4] and implemented in the Limonade[3] microframework. This microframework offers a number of functions that facilitate the implementation of services based on HTTP(S) protocol methods.

One of the most important functionalities of the web service is searching for namespaces for a specific prefix. To receive data, the `GET` method should be used. When a client calls the service, documents containing the specified prefix are retrieved from the database. If there is no namespace, the response code is `404`. The default mediatype is JSON-LD and the second type is Turtle. To receive Turtle data the `Accept` header with the `text/turtle` value should be used. A response return 15 items by default. The `limit` parameter after query sting is used to change the number of displayed elements.

[3] `https://limonade-php.github.io/`

Our tool supports namespace declarations for Turtle, RDF/XML, RDFa, JSON-LD serializations, SPARQL query language, CSV and TSV data formats. To receive a declaration in these formats the `GET` method sould be used. The returned data can be in plain text or in Turtle.

Being signed in a user receive the access to perform additional operations. For this purpose, the OAuth2 [5,6] is used to authenticate users, which allows to identify users using well-known OAuth2 servers such as Google, Facebook, Twitter, Github, etc. To perform the operations that required for authentication, the `Authorization` header with the `Bearer access-token` value should be sent. An authenticated user can add new namespaces to a given prefix. To successfully add a namespace, user should send a JSON-LD document by using the `POST` method.

The second operation requiring authentication is voting. Having a namespace, we can vote 'for' (plus) or 'against' (minus) using the `POST` method. In one prefix we have the possibility to add one plus and many minuses. If one make a mistake, she/he can delete the voice by using the `DELETE` method. If the `POST` method is executed successfully the response `201` code is returned, and if the resource is already exists the `409` code is returned.

Another feature of the web service is the ability to preview popular prefixes that have the most votes. To receive the answer, a user should send `GET` method. The response can be in the JSON-LD or in Turtle along with the pagination of the Hydra vocabulary [7]. In this URL user can set two parameters: `offset` and `page`. The first parameter defines how many prefixes can be displayed per a page, while the second parameter indicate the current page.

The key function of our tool is to modify the Turtle's documents. This operation finds prefixes that do not have declared namespaces and choose the best one. Finally, the document is displayed in the body of the response. The supported formats are Turtle, JSON-LD, N-Triples, and RDF/XML.

2.3 Turtle$_d$ and Console Application

In this subsection we propose Turtle$_d$. It is a serialization that extends Turtle to support many decentralized RESTful web services. We expand the Turtle grammar to the `@prefix_endpoint` directive, which points to various web services that resolve prefixes. Listing 1.1 shows an extended grammar of Turtle in EBNF.

```
[3]  directive ::= prefixID | base | sparqlPrefix
                 | sparqlBase | endpoint
[5e] endpoint  ::= '@prefix_endpoint' IRIREF '.'
```

Listing 1.1. Turtle$_d$ grammar

Another independent part of our architecture is the Bash script, which aims to show the use of the web service (see Subsection 2.2). This console application works according to the following steps. In the first step, the script finds the `@prefix_endpoint` directive in the Turtle/Turtle$_d$ document that contains the web service IRI references. As a result, the web service returns a response about

successful or negative modification of the Turtle document. If the script receive a negative response, it attempt to send to the other existing web service form the second `@prefix_endpoint` directive.

2.4 Web Application

A web application is designed to support the web service (Subsection 2.2), which is designed to send HTTP requests and present data in a user-friendly form. Each webpage is described using RDFa. The application is adapted to mobile devices. The elements on the webpages have been separated so that a user can click exactly on one element in the mobile devices.

The application at the top has a menu to help you navigate the site. In the upper right corner there is the *Sign in to get permission* button which allows users to sign in. Being signed in, a user get permission to manage namespaces. On the main page of the site there is also a search engine that helps to find a specific prefix.

After entering the prefix, the middle part of the page is reloaded. Below the menu there is a header that inform about the current prefix. A user can vote for the namespace with plus or minus by clicking the appropriate button. When a user click *Add alternative namespace*, a text box appears and let a user to add a new namespace. At the bottom of the page there are a few data formats with current prefix and namespace.

The web application also can show the popular prefixes. The key feature of our application is the preparation of Turtle$_d$ documents (see Subsection 3).

3 Demonstration

The RESTful web service can be tested at `https://prefix.chemskos.com/api/?/`. One of the main functionalities is finding namespaces of prefix. This operation requires `GET` method. Listing 1.2 shows the namespaces related to prefixes.

```
curl -X GET 'https://prefix.chemskos.com/api/foaf'\
-H 'accept: text/turtle'
```

Listing 1.2. Request in `curl` showing namespaces

The obtained data can be in JSON-LD or Turtle formats. If there are not any prefixes, it will be returned `404` error. Listing 1.3 shows response containing namespaces in Turtle format.

```
@prefix pres: <http://ii.uwb.edu.pl/prefix_resolver#> .
[] <pres:namespaces> (
   [
     <pres:namespace> "http://xmlns.com/foaf/0.1/" ;
     <pres:score> 1 ;
   ]
   ) .
```

Listing 1.3. Response showing namespaces

The response to access the namespace declaration is shown in Listing 1.4. The supported formats are Turtle, RDF/XML, RDFa, JSON-LD, SPARQL, CSV, and TSV.

```
curl -X GET \
'https://prefix.chemskos.com/api/?/foaf/formats/rdf/xml' \
-H 'accept: text/plain'
```

Listing 1.4. Request in `curl` for declaration namespace

Namespaces are added using `POST` method. The active access token is neccessary to perform function. The request has been shown on Listing 1.5.

```
curl -X POST \
'https://prefix.chemskos.com/api/foaf' \
-H 'accept: application/ld+json' \
-H 'Authorization: Bearer access-token' \
-d '{
"@context": {"pres": "http://ii.uwb.edu.pl/prefix_resolver
    #"},
"@id": "https://prefix.chemskos.com/api/foaf",
"pres:namespace": "http://namespace.com/ns#"
}'
```

Listing 1.5. Request in `curl` for creating a new namespace

Displaying popular prefixes can be presented in JSON-LD or Turtle serialziations. The request has been shown shown on Listing 1.6

```
curl -X GET \
'https://prefix.chemskos.com/api/?/popular' \
-H 'accept: text/turtle' \
```

Listing 1.6. Request in `curl` showing popular prefixes

The last functionality is submitting Turtle$_d$ or Turtle documents. After modification the RDF file can be downloaded. Listing 1.7 shows the submission of Turtle$_d$. After transformation N-Triples is expected.

```
curl -X POST \
'https://prefix.chemskos.com/api/prepare' \
-H 'accept: application/n-triples' \
-F file=@path_to_file.ttl
```

Listing 1.7. Submitting Turtle$_d$ document

The source code can be found on https://github.com/KarolLitman/Dictionary-of-RDF-Prefixes-webservice.

The API documentation was created, which facilitates the understanding of the service in a readable manner. The documentation consists of queries that can be performed on the RESTful web service. Additionally, by clicking on a given method, a user obtains sample source code, structure of the query,

required parameters and displaying all possible errors in the method. Swagger[4] was used to create the API documentation. It can be found on `https://prefix.chemskos.com/api/doc/`.

The web application can be viewed at `https://prefix.chemskos.com/`. The application at the top has a menu to help users navigate the site. The menu contains hyperlinks for the *Home* page, *Popular* page and *Prepare document* page. In the right corner there is the *Sign in to get permission* button which allows a user to sign in. In Fig. 3 the web page with found prefix is presented.

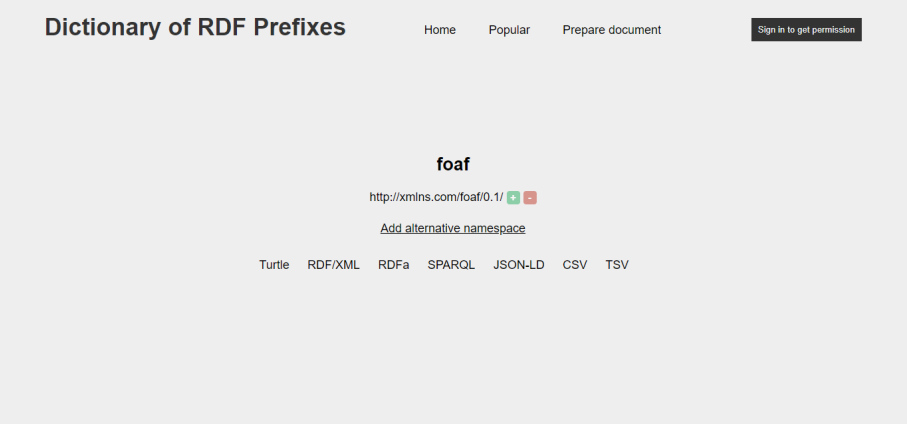

Fig. 3. Web page with found prefix

The use of the web application has been presented on the screencast and published on **https://www.youtube.com/watch?v=-hSYA8epMlA**. The source code can be found on https://github.com/KarolLitman/Dictionary-of-RDF-Prefixes-webapp.

The use of the console application has been presented on the screencast and published on **https://www.youtube.com/watch?v=O4tjuo1BB24**. The screencast shows how the application deals with broken endpoints. The source code can be found on https://github.com/KarolLitman/Dictionary-of-RDF-Prefixes-bashscript.

4 Conclusions

Decentralization is the key to resilience in Internet applications. In this paper we present DRPD tools that are decentralized. We demonstrate four elements of our architecture: RESTful document-oriented database, RESTful web service with documented API, web application with client-side that remote calling

[4] `http://docs.swagger.io/spec.html`

resources from our endpoint, and console application that also use our web service. Moreover, we propose Turtle$_d$ serialization that supports endpoints that are compatible with our architecture.

We have taken an initial step in easier delivery of prefixes and namespaces. In future work we would like to extend this work to other serializations that use prefixes.

References

1. Ben Adida, Shane McCarron, Ivan Herman, and Mark Birbeck. RDFa Core 1.1 - Third Edition. W3C recommendation, W3C, March 2015. http://www.w3.org/TR/2015/REC-rdfa-core-20150317/.
2. J Chris Anderson, Jan Lehnardt, and Noah Slater. *CouchDB: The Definitive Guide: Time to Relax*. O'Reilly Media, Inc., 2010.
3. Tim Bray, Andrew Layman, Richard Tobin, and Dave Hollander. Namespaces in XML 1.1 (Second Edition). W3C recommendation, W3C, August 2006. http://www.w3.org/TR/2006/REC-xml-names11-20060816/.
4. Roy T Fielding and Richard N Taylor. *Architectural styles and the design of network-based software architectures*, volume 7. University of California, Irvine Doctoral dissertation, 2000.
5. D. Hardt. The OAuth 2.0 Authorization Framework. RFC 6749, RFC Editor, October 2012. http://www.rfc-editor.org/rfc/rfc6749.txt.
6. M. Jones and D. Hardt. The OAuth 2.0 Authorization Framework: Bearer Token Usage. RFC 6750, RFC Editor, October 2012. http://www.rfc-editor.org/rfc/rfc6750.txt.
7. Markus Lanthaler, Gregg Kellogg, and Manu Sporny. JSON-LD 1.0. W3C recommendation, W3C, January 2014. http://www.w3.org/TR/2014/REC-json-ld-20140116/.
8. Eric Prud'hommeaux and Gavin Carothers. RDF 1.1 Turtle. W3C recommendation, W3C, February 2014. http://www.w3.org/TR/2014/REC-turtle-20140225/.
9. Guus Schreiber and Fabien Gandon. RDF 1.1 XML Syntax. W3C recommendation, W3C, February 2014. http://www.w3.org/TR/2014/REC-rdf-syntax-grammar-20140225/.
10. Manu Sporny, Ivan Herman, Ben Adida, and Mark Birbeck. RDFa 1.1 Primer - Third Edition. W3C note, W3C, March 2015. http://www.w3.org/TR/2015/NOTE-rdfa-primer-20150317/.