Phuzzy.link: A SPARQL-powered Client-Sided Extensible Semantic Web Browser

Blake Regalia, Krzysztof Janowicz and Gengchen Mai

STKO Lab, University of California, Santa Barbara, USA

Abstract. With a rapidly increasing amount of machine-readable resources being generated and published to the Linked Data cloud, human-readable representations of those resources continue to serve an important demand. While dereferencing interfaces offer an easy, general-purpose solution to providing human-readable representations, they are not always well-equipped to support the specific characteristics of a dataset, such as having the ability to render embedded maps for displaying geometries. Another shortcoming to exploring resources via dereferencing interfaces comes from deciding how to handle nodes that belong to remote IRIs. On one hand, a user may wish to leave the current data source and dereference the remote IRI. On the other hand, the user may wish to remain within the current interface and view the dataset's own triples about the given node. This leads to an interesting dilemma since any such human-readable representation of a remote IRI is no longer the result of dereferencing. Finally and perhaps most importantly, many data providers do not deploy and maintain dereferencing interfaces at all. In this paper, we address these issues by presenting a fully-extensible Web interface for exploring remote RDF datasets by querying their SPARQL endpoints from the client.

1 Introduction and Motivation

The rapid increase in graphical interfaces for browsing, querying, visualizing, mining, linking, and summarizing Linked Data and ontologies highlights their increasing complexity and heterogeneity as well as the interest of a broader community. Each of these interfaces either addresses a certain gap in previous work, rests on a different stack of technologies, serves a particular domain, or provides a visual interface for a new workflow or tasks, such as co-reference resolution or ontology alignment [13]. While this wealth of interfaces may seem redundant and makes the selection of a particular interface more challenging to the end user, it enables the community to test new ideas and converge on useful features, while unsuccessful approaches disappear over time. One family of such interfaces is concerned with providing a frontend for SPARQL endpoints, with Pubby [4] being one of the first and most popular systems. Another family are Linked Data browsers that enable humans to navigate the Linked Data cloud, Tabulator being the first widely used implementation [2]. Today we are seeing interfaces that combine a variety of functionalities such as query construction and result visualization as well as those that try to address the problem that the heterogeneity of datasets, interfaces, frontends, endpoints, and so forth, makes it difficult to ensure a consistent user experience across the Linked Data cloud. For instance, a client-side browsing interface should put the user in charge of deciding how to render Linked Data from distant SPARQL endpoints.

Motivated by these challenges, we developed Phuzzy.link[1], a Semantic Web Browser that gives data publishers the ability to adapt the interface to support special characteristics of their datasets, yet also puts users in full control over their viewing experience. We try to address issues that we outline about existing interfaces in the Related Work section. **To summarize, the novel contributions of our approach are as follows:**

- A public interface that is free, open-source and easily extensible by allowing anyone to develop and integrate their own publishable plug-ins.
- The system operates from entirely within the browser which means that (1) no installation is required by hosts nor clients and (2) all content are fetched directly from the target endpoint rather than through some middleman.
- The hosts of endpoints (and ostensibly the data publishers themselves) have exclusive access to authoritatively set and modify the default configurations used when browsing their endpoint URL.

In this work, we present the interface, demonstrate its usage with an example from a DBpedia resource, and showcase how plug-ins can enhance the user experience based on arbitrary ontological predicates within the context of a given dataset.

2 Related Work

There are several Semantic Web browsers available[2] to the community, but each of these applications are limited by general-purpose usability. We see an opportunity to provide a general-purpose browser that is more sustainable by offering open extensibility of its content selection and content formatting. In this section, we limit ourselves to review a few selected systems, as a recent, up-to-date survey has been provided by [5].

Pubby [4] is a popular general-purpose Linked Data frontend that can be deployed atop a SPARQL endpoint. Pubby runs as a Java servlet, meaning that it must first be installed and then configured on a per-endpoint basis. Our interface works entirely in the browser, requires no installation, and is ready to browse any public SPARQL endpoint that allows Cross-Origin Resource Sharing (CORS). Furthermore, Pubby only works with SPARQL endpoints that support DESCRIBE queries, while our service exclusively relies on SELECT queries to download triples.

[1]The system and its documentation can be found at `http://phuzzy.link`, and the source code and plug-ins at `https://github.com/blake-regalia/phuzzy.link`.

[2]`https://www.w3.org/2001/sw/wiki/Category:Semantic_Web_Browser`

LodLive [3] offers an RDF resource browser as an interactive graph visualization that dynamically pulls information from the Web using Linked Data standards such as SPARQL and RDF (via content-negotiation). Another of their team's product is **LodView**[3], which presents a static tabular display for each RDF resource. Much like our approach, these tools are capable of dynamically constructing human-readable representations of RDF resources, and at times entirely from the client. The problem we sought to address in our interface however is with the limitation imposed by prescribed interface components and the inherent variability of datatypes. Where our interface stands apart is how we give users full control over content selection and content formatting of the display, as well as the styling and function of the interface itself.

RelFinder [8] offers an interactive visualization of relationships between resources by querying SPARQL endpoints from the client. RelFinder and our interface are mechanically similar in how they are able to query any remote endpoint. However, Phuzzy.link is intended for creating low-level, human-readable representations of individual resources in the context of some given triplestore.

Fresnel [11] is an RDF vocabulary designed to universally tackle the problems of *content selection* and *content formatting* for the presentation of human-readable RDF content. Since Fresnel is geared towards enabling high-level representations of resources, it also allows for 'less relevant' triples and identifiers to be obscured from the resulting interface. Our intention with Phuzzy.link is to show a comprehensive overview of the underlying data without sacrificing any of the *hard* RDF information. Setting these differences aside, Fresnel may still be a suitable vocabulary for our interface and we hope to add support for it in the future.

Sextant [1] is a web application that explores linked geospatial datasets by presenting interactive maps with advanced content filtering. It is primarily designed to function as a map-based Geographic Information System. Similar to our approach, it is capable of importing map content by querying SPARQL endpoints from the client. However, getting Sextant set up to use some arbitrary dataset can be quite involved as it may require special configuration for an unmapped ontology or unexpected predicates and datatypes. Our interface ships ready to deploy on any SPARQL endpoint without the need for additional configurations, yet also provides simple means to override configurations for ad-hoc browsing sessions.

Uduvudu [9] is an adaptive UI engine for Linked Data which demonstrates its flexibility by promoting the design of reusable design components for Linked Data user interfaces. Whereas the Uduvudu framework does not provide support for extensions to reason on input data, our system enables plug-ins to bind to arbitrary triple-related events, e.g., responding to triples grouped by common predicates, object datatypes, language tags of literals, and so on. Furthermore, Uduvudu loads content into a viewing frame while the browser page itself remains at a static URL. Our system acts as a sort of third-party pseudo-dereferencing interface by constructing the human-readable representation of a resource from

[3]`http://lodview.it/`

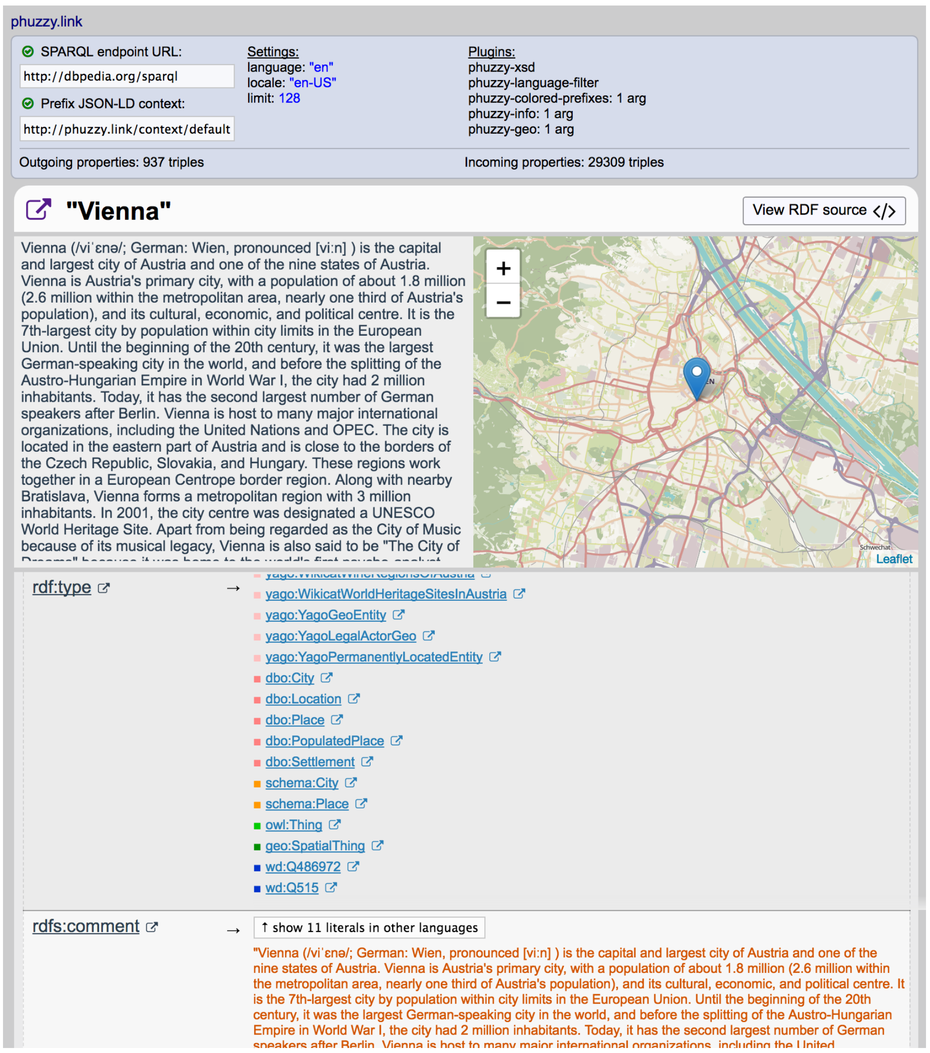
the request URL alone, which contains the location of the target SPARQL endpoint and the URI of the subject node to query.

3 Interface Design and Functionality

The nature of datasets across today's Linked Data cloud is diverse and unpredictable. We believe there is a widespread consensus that many of the Web's current general-purpose, plug-and-play dereferencing interfaces are limited by the seemingly inherent *one size does not fit all* problem. In our approach, we designed a system that tries to embrace the *one size does not fit all* problem by enabling the community to contribute their own plug-ins that extend the user interface with special styling, settings, and interactive components. In fact, there is no limitation as to what a plug-in can do once it is active within the interface, e.g., creating interactive maps, rendering 3D visualizations, making auxilliary HTTP requests, and so on. We believe that this feature alone makes our interface highly adaptable to the content of any RDF dataset.

3.1 Initialization

In order to best describe how Phuzzy.link constructs a human-readable representation of a Linked Data resource, we explain each step of the process by following a contrived scenario. Suppose a user's intention is to explore DBpedia, starting with the resource for Vienna, the resource `http://dbpedia.org/resource/Vienna`. We begin by constructing a URL in the format given by the simple API documentation shown in Table 1 for the `/browse` action. For the parameters **endpoint** and **resource**, we plug in the URL for DBpedia's SPARQL endpoint `http://dbpedia.org/sparql` and the URI for the resource we are interested in. This yields the URL `http://phuzzy.link/browse/http://dbpedia.org/sparql#<http://dbpedia.org/resource/Vienna>` which when given a GET request, responds with an HTTP 301 redirect to `http://phuzzy.link/browse/dbpedia.org/sparql#<http://dbpedia.org/resource/Vienna>`, which has removed the origin portion 'http://' from the endpoint argument of the URL path for clarity. The subsequent GET request receives an HTML document response from the server, which eventually loads the necessary interface assets such as its scripts, configurations, and stylesheets. At that point, no other information has to be exchanged between the client and the Phuzzy.link server. All information about the resource will be fetched directly from the SPARQL endpoint by the client via HTTP requests. For this reason, the part of the URL that identifies which resource to load is contained within the fragment identifier, e.g., '`#<http://dbpedia.org/resource/Vienna>`'. This ensures that subsequent changes to the fragment identifier, e.g., by following a link to another node within the dataset, does not trigger additional HTTP requests to the Phuzzy.link server. In fact, by the time a resource has finished loading in the interface, the client's document already contains all the assets it needs from Phuzzy.link to continue browsing the same dataset.



phuzzy.link

SPARQL endpoint URL: http://dbpedia.org/sparql

Prefix JSON-LD context: http://phuzzy.link/context/default

Settings:
language: "en"
locale: "en-US"
limit: 128

Plugins:
phuzzy-xsd
phuzzy-language-filter
phuzzy-colored-prefixes: 1 arg
phuzzy-info: 1 arg
phuzzy-geo: 1 arg

Outgoing properties: 937 triples

Incoming properties: 29309 triples

"Vienna"

View RDF source </>

Vienna (/viˈɛnə/; German: Wien, pronounced [viːn]) is the capital and largest city of Austria and one of the nine states of Austria. Vienna is Austria's primary city, with a population of about 1.8 million (2.6 million within the metropolitan area, nearly one third of Austria's population), and its cultural, economic, and political centre. It is the 7th-largest city by population within city limits in the European Union. Until the beginning of the 20th century, it was the largest German-speaking city in the world, and before the splitting of the Austro-Hungarian Empire in World War I, the city had 2 million inhabitants. Today, it has the second largest number of German speakers after Berlin. Vienna is host to many major international organizations, including the United Nations and OPEC. The city is located in the eastern part of Austria and is close to the borders of the Czech Republic, Slovakia, and Hungary. These regions work together in a European Centrope border region. Along with nearby Bratislava, Vienna forms a metropolitan region with 3 million inhabitants. In 2001, the city centre was designated a UNESCO World Heritage Site. Apart from being regarded as the City of Music because of its musical legacy, Vienna is also said to be "The City of

rdf:type →
- yago:WikicatWineRegionsOfAustria
- yago:WikicatWorldHeritageSitesInAustria
- yago:YagoGeoEntity
- yago:YagoLegalActorGeo
- yago:YagoPermanentlyLocatedEntity
- dbo:City
- dbo:Location
- dbo:Place
- dbo:PopulatedPlace
- dbo:Settlement
- schema:City
- schema:Place
- owl:Thing
- geo:SpatialThing
- wd:Q486972
- wd:Q515

rdfs:comment → ↑ show 11 literals in other languages

"Vienna (/viˈɛnə/; German: Wien, pronounced [viːn]) is the capital and largest city of Austria and one of the nine states of Austria. Vienna is Austria's primary city, with a population of about 1.8 million (2.6 million within the metropolitan area, nearly one third of Austria's population), and its cultural, economic, and political centre. It is the 7th-largest city by population within city limits in the European Union. Until the beginning of the 20th century, it was the largest German-speaking city in the world, and before the splitting of the Austro-Hungarian Empire in World War I, the city had 2 million inhabitants. Today, it has the second largest number of German speakers after Berlin. Vienna is host to many major international organizations, including the United

Fig. 1 The interface view for `dbr:Vienna`. At the very top, we see the configuration, settings, and plug-ins being used for this session. Below that, information about the resource is listed for the rest of the document. The text and map box near the top of this section represent the *abstract* for this resource, determined by the plug-ins `phuzzy-info` and `phuzzy-geo`, which have been initialized with arguments that dictate which predicates to use for obtaining the summary text and geographic coordinates, respectively.

Method, URL and Description
`GET /browse/:endpoint#:resource` Loads an interface that will ultimately query the given **endpoint** URL (using the HTTP protocol by default) for triples about the given **resource** (either a prefixed name or an absolute IRI enclosed by angle brackets `< >`)
`GET /config/:endpoint` Fetch the server's JSON config file associated with the given **endpoint**
`PUT /config/:endpoint` Replace the server's JSON config file for the given **endpoint**. This request must originate from an IP listed in the DNS A or AAAA records for the given **endpoint**'s hostname
`GET /plugin/:name@:version` `Accept: application/json` Fetch the asset manifest for the plugin given by its npm **name** and **version** (npm is a package manager for JavaScript). The asset manifest is sourced from its `package.json` file
`GET /plugin/:name@:version` `Accept: text/html,*` Describe the status of this plugin, i.e., whether or not it is installed on the server and ready to be loaded in an interface. This is also where a user can request the server install a given plugin
`GET /plugin/:name@:version/:asset` Fetch the **asset** file contained within the given plugin

Table 1 The HTTP API for interacting with `http://phuzzy.link/`

Once the DOMContentLoaded[4] event is fired, the interface script parses the current URL and then starts fetching all the assets it will need to browse an endpoint. For our DBpedia Vienna scenario, the endpoint URL is given by 'dbpedia.org/sparql' so the script will GET request '`/config/dbpedia.org/sparql`' to download a JSON config file that describes several options to use when browsing this endpoint including: the URL for a JSON-LD file containing a '`@context`' key for the map of prefixes to use when shortening and expanding IRIs, a list of plug-ins (identified by their NPM name and version; see 'Plug-in Support' in Section 3.2) to load in this version of the interface along with the initialization arguments for instantiating each plug-in, and options for controlling the behavior of the SPARQL requests such as the default LIMIT to use per SPARQL query when loading triples in multiple chunks. An example for our contrived DBpedia configuration is shown in Listing 1. Although we created a default JSON-LD context for this example, any endpoint that does not explicitly have its own configuration inherits the default prefix.cc[5] JSON-LD context as fallback. Once the map of prefixes is loaded, the script may automatically change the hash fragment identifier by shortening the IRI to a prefixed name, so the final URL our browser resolves for the Vienna example is `http://phuzzy.link/browse/dbpedia.org/sparql#dbr:Vienna`.

[4]`https://developer.mozilla.org/en-US/docs/Web/Events/DOMContentLoaded`

[5]`http://prefix.cc/context`

```
{
    "context": "http://phuzzy.link/context/default",
    "order": ["rdf:*", "rdfs:*", "dbo:*", "dbp:*"],
    "settings": {
        "language": "en",
        "locale": "en-US",
        "limit": 128
    },
    "plugins": [
        {"name": "phuzzy-xsd", "version": "^0.0.1"},
        {"name": "phuzzy-language-filter", "version": "^0.0.1"},
        {"name": "phuzzy-colored-prefixes", "version": "^0.0.1", "args": {"palettes": 6}},
        {"name": "phuzzy-info", "version": "^0.0.1", "args": {
            "predicates": ["http://dbpedia.org/ontology/abstract"]
        }},
        {"name": "phuzzy-geo", "version": "^0.0.1", "args": {
            "wkt_predicates": ["http://www.w3.org/2003/01/geo/wgs84_pos#geometry"]
        }}
    ]
}
```

Listing 1 Our contrived JSON config for DBpedia, available at `http://phuzzy.link/config/dbpedia.org/sparql`

Only after the prefixes have been loaded and all plug-ins have been initialized, the interface begins to query for both outgoing and incoming triples that belong to the given node. As SPARQL results arrive to the client, our Web application constructs the interface's document elements dynamically. During this loading phase, plug-ins that have bound to certain triple-related events will be fired so that they may mutate elements with custom styling and interaction responses. For example, the `phuzzy-geo` plug-in binds to the predicate event for '`http://www.w3.org/2003/01/geo/wgs84_pos#geometry`' so that it may intercept the triple's geometry string literal and render a supplemental map view in the summary box at the top of the page.

3.2 Plug-in Support

The initial release of our interface supports plug-ins that have been uploaded to the public NPM[6] repository. This technique simplifies plug-in management since versioning, namespacing, package manifest/details, and hosting are all handled by the repository. Once a plug-in is ready for adoption, the Phuzzy.link server needs to 'install' its package so that its frontend assets can be bundled, such as scripts and stylesheets. A user can trigger this action by using the HTTP API shown in Table 1.

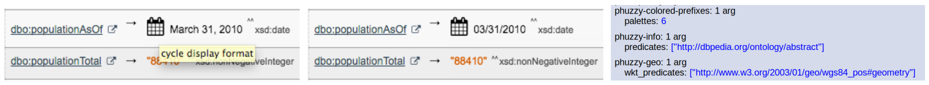
Plug-ins work by binding callback handlers to special event types. In essence, plug-ins can choose to be triggered for triples that contain specific predicates, term types, datatypes, and so forth. This approach allows plug-ins to remain simple and focus solely on code that is specific to their stlying and interaction.

To give an example of how a plug-in can be used and its effect on the user experience, `phuzzy-geo` renders static geospatial data from the current resource onto a map display. As seen in Figure 1, this plug-in has interpreted the point geometry for the location of `dbr:Vienna` and plotted it to the map. In this example, the configuration being used to browse DBpedia (see Listing 1) passes an

[6]Node Package Manager - https://www.npmjs.com

argument to the **`phuzzy-geo`** plug-in that specifies which predicates to search for Well-Known Text strings, in this case it binds to the `w3cgeo:geometry` predicate.

Other plug-ins include the aforementioned **`phuzzy-info`** to load and style data from abstracts, **`phuzzy-xsd`** for XSD datatypes, e.g., to cycle through different representation of dates, **`phuzzy-language-filter`** to filter content by language, and **`phuzzy-colored-prefixes`** to color-code prefixes; see Fig 2.


dbo:populationAsOf → March 31, 2010 ^^xsd:date
dbo:populationTotal → "88410" ^^xsd:nonNegativeInteger

dbo:populationAsOf → 03/31/2010 ^^xsd:date
dbo:populationTotal → "88410" ^^xsd:nonNegativeInteger

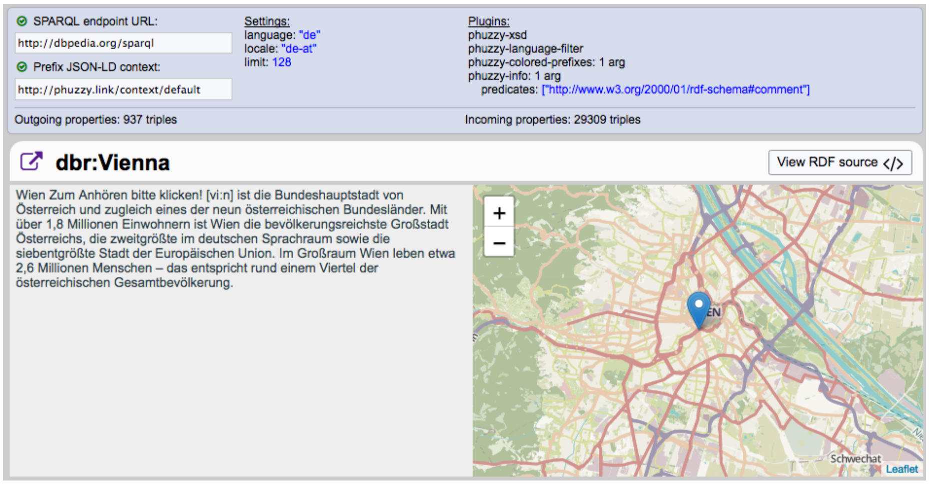
phuzzy-colored-prefixes: 1 arg
    palettes: 6
phuzzy-info: 1 arg
    predicates: ["http://dbpedia.org/ontology/abstract"]
phuzzy-geo: 1 arg
    wkt_predicates: ["http://www.w3.org/2003/01/geo/wgs84_pos#geometry"]

Fig. 2 The figures show a date conversion example using the **`phuzzy-xsd`** plug-in as well as the dataset specific settings for the other activated plug-ins.

3.3 Overriding the Defaults

We believe that users should feel empowered by our interface rather than limited by its assumptions. For example, if a user is browsing a dataset that has loaded a prescribed configuration, there may be some options the user would prefer to modify during their session. We provide a list of query string parameters in Table 2 that allow the user to override configuration options, tune interface settings, load new plug-ins, and change arguments to existing plug-ins.

For instance, we revisit our previous **`dbr:Vienna`** example and override the language, locale and an argument to the **`phuzzy-info`** plug-in by adding the query string arguments `?language=en` `&locale=de-at` and a URI encoded JSON string for `&plugin.phuzzy-info="..."`. The resulting interface display is shown in Figure 3 and is available at `http://bit.ly/2v4x8xK`.


SPARQL endpoint URL:
http://dbpedia.org/sparql
Prefix JSON-LD context:
http://phuzzy.link/context/default

Settings:
language: "de"
locale: "de-at"
limit: 128

Plugins:
phuzzy-xsd
phuzzy-language-filter
phuzzy-colored-prefixes: 1 arg
phuzzy-info: 1 arg
    predicates: ["http://www.w3.org/2000/01/rdf-schema#comment"]

Outgoing properties: 937 triples
Incoming properties: 29309 triples

dbr:Vienna

Wien Zum Anhören bitte klicken! [vi:n] ist die Bundeshauptstadt von Österreich und zugleich eines der neun österreichischen Bundesländer. Mit über 1,8 Millionen Einwohnern ist Wien die bevölkerungsreichste Großstadt Österreichs, die zweitgrößte im deutschen Sprachraum sowie die siebentgrößte Stadt der Europäischen Union. Im Großraum Wien leben etwa 2,6 Millionen Menschen – das entspricht rund einem Viertel der österreichischen Gesamtbevölkerung.

Fig. 3 The interface display for **`dbr:Vienna`** after overriding default configuration options and settings with the German language, German-Austrian locale, and using **`rdfs:comment`** for the abstract text instead of **`dbo:abstract`**. `http://bit.ly/2v4x8xK`

Parameter key	**Argument value description**
language	A BCP 47 language tag that dictates which language to prefer, e.g., when selecting literals to display more prominently.
locale	A BCP 47 language tag that dictates which locale to prefer when formatting dates, times, and so forth.
limit	The number of triples to limit in the response of each SPARQL query.
context	The URL of a JSON-LD file that defines the map of prefixes to use when shortening IRIs.
plugin.*plugin_name*	A JSON string to pass as the argument when initializing the given plugin.

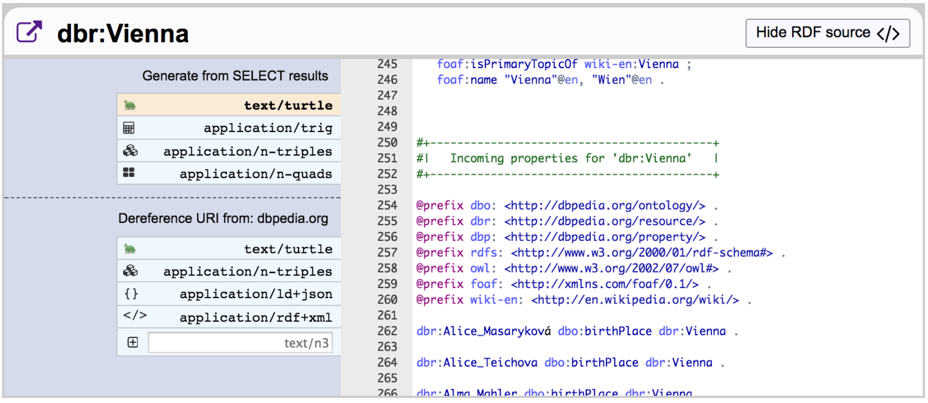
Table 2 A few query string parameters that allow users to override configurations and settings for the `/browse` action.

3.4 Browsing and Dereferencing

Similar to systems such as Pubby, our interface displays a resource by showing its outgoing and incoming properties, i.e., triples that contain the resource IRI in the subject or object position, in a tabular display that takes the main focus of the interface body. However, the paradigm we establish in this interface is that the client is exploring a *dataset*, not just an independent resource. Therefore, all IRIs, including predicates and nodes, are rendered as local hyperlinks (using the URL fragment identifier) such that following a node's link will load that resource in the same interface from the current dataset. However, this approach by itself would sacrifice the expectation that a hyperlinked RDF node should resolve to its dereferenceable URI. Instead, we offer both options to the client. Next to each IRI, there is also an *external link* icon that points to the node's actual IRI, allowing the client to dereference the resource in a new tab.

In order to help salient properties be better staged near the top, the owner of a dataset, or the client itself, may define the order that rows of triples are sorted in based on predicate IRIs and predicate patterns. For example, we find `rdf:type` should be the first triples shown when browsing *our* dataset, followed immediately by any other rdf: properties, then rdfs: properties, and so on. Our prescribed sort order might look something like this: `["rdf:type","rdf:*","rdfs:*",...]`. See Listing 1 for a DBpedia example.

Users who find the interface convenient for browsing an RDF dataset may eventually wish to access a resource's underlying RDF code. Theoretically, they should be able to dereference the URI and with a bit of content-negotiation acquire the resource in some RDF serialization format. However, this may require manually setting headers, leaving the browser to use another networking tool such as cURL, or perhaps their serialization format of choice is simply not available. Furthermore, dereferencing a node that is external to an RDF dataset implies that the resulting triples do not reflect the endpoint's own RDF dataset. Our solution to this situation is a display toggle that allows end-users to view the current resource in an RDF serialization format of their choice. As we show



```
    foaf:isPrimaryTopicOf wiki-en:Vienna ;
    foaf:name "Vienna"@en, "Wien"@en .



#+------------------------------------------+
#|   Incoming properties for 'dbr:Vienna'   |
#+------------------------------------------+

@prefix dbo: <http://dbpedia.org/ontology/> .
@prefix dbr: <http://dbpedia.org/resource/> .
@prefix dbp: <http://dbpedia.org/property/> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix foaf: <http://xmlns.com/foaf/0.1/> .
@prefix wiki-en: <http://en.wikipedia.org/wiki/> .

dbr:Alice_Masaryková dbo:birthPlace dbr:Vienna .

dbr:Alice_Teichova dbo:birthPlace dbr:Vienna .

dbr:Alma_Mahler dbo:birthPlace dbr:Vienna
```

Fig. 4 The *RDF source* display of our interface that generates RDF from SPARQL SELECT results, and enables dereferencing with prescribed MIME types.

in Figure 4, serialized text is dynamically generated from already downloaded SPARQL SELECT results, ensuring that the final output is an accurate reflection of the triples contained by the encapsulated dataset. This also essentially bypasses the need to make additional requests for DESCRIBE queries which are not always supported by SPARQL endpoints. However, we also complement this feature by enabling the client to dereference a resource using some RDF serialization format. The client may select or manually enter a MIME type that will set the `Accept` header of the HTTP request to the resource URI.

4 Conclusions

In this paper, we presented Phuzzy.link, an extensible Semantic Web browser that functions entirely on the client-side and within the browser to access remote RDF datasets via their public SPARQL endpoints. Unlike other Semantic Web browsers, our interface is designed to be adaptable with generic configurations, e.g., provided by data publishers, local settings that override these configurations, and plug-ins that give users control over content selection and content formatting. So far, these plug-ins work by responding to simple triggers such as predicates, e.g., `dbo:abstract`, datatypes, or more complex situations such as blank nodes that point to geometries. In the future, we plan to identify and use ontology design patterns [6]. For instance, the presence of the trajectory pattern [7] could trigger the interface to connect the individual point locations (lat/long pairs) by lines, thereby interpolating a path. It could also display temporal data or load a time-slider [12]. Consequently, data released using ontologies that make use of certain patterns can be explored in custom ways; see also [10]. We hope that a community will form around the plug-in-design-pattern interface to contribute to Phuzzy.link.

Acknowledgements

Partial funding has been provided by USGS under the *Linked Data for the National Map* award as well as by NSF under the awards 1440139 (GeoLink) and 1540849 (X-DOMES).

References

1. Bereta, K., Nikolaou, C., Karpathiotakis, M., Kyzirakos, K., Koubarakis, M.: Sextant: Visualizing time-evolving linked geospatial data. In: Proceedings of the 2013th International Conference on Posters & Demonstrations Track-Volume 1035, pp. 177–180. CEUR-WS. org (2013)
2. Berners-Lee, T., Chen, Y., Chilton, L., Connolly, D., Dhanaraj, R., Hollenbach, J., Lerer, A., Sheets, D.: Tabulator: Exploring and analyzing linked data on the semantic web. In: Proceedings of the 3rd international semantic web user interaction workshop, vol. 2006, p. 159. Athens, Georgia (2006)
3. Camarda, D.V., Mazzini, S., Antonuccio, A.: Lodlive, exploring the web of data. In: Proceedings of the 8th International Conference on Semantic Systems, pp. 197–200. ACM (2012)
4. Cyganiak, R., Bizer, C.: Pubby-a linked data frontend for sparql endpoints. Url: http://wifo5-03.informatik.uni-mannheim.de/pubby/ (Accessed: 07-26-2017) (2007)
5. Dadzie, A.S., Pietriga, E.: Visualisation of linked data–reprise. Semantic Web **8**(1), 1–21 (2017)
6. Gangemi, A., Presutti, V.: Ontology design patterns. In: Handbook on ontologies, pp. 221–243. Springer (2009)
7. Hu, Y., Janowicz, K., Carral, D., Scheider, S., Kuhn, W., Berg-Cross, G., Hitzler, P., Dean, M., Kolas, D.: A geo-ontology design pattern for semantic trajectories. In: International Conference on Spatial Information Theory, pp. 438–456. Springer (2013)
8. Lohmann, S., Heim, P., Stegemann, T., Ziegler, J.: The relfinder user interface: interactive exploration of relationships between objects of interest. In: Proceedings of the 15th international conference on Intelligent user interfaces, pp. 421–422. ACM (2010)
9. Luggen, M., Gschwend, A., Anrig, B., Cudré-Mauroux, P.: Uduvudu: a graph-aware and adaptive ui engine for linked data. In: LDOW@ WWW (2015)
10. Musetti, A., Nuzzolese, A.G., Draicchio, F., Presutti, V., Blomqvist, E., Gangemi, A., Ciancarini, P.: Aemoo: Exploratory search based on knowledge patterns over the semantic web. Semantic Web Challenge **136** (2012)
11. Pietriga, E., Bizer, C., Karger, D., Lee, R.: Fresnel: A browser-independent presentation vocabulary for rdf. In: International Semantic Web Conference, vol. 4273, pp. 158–171. Springer (2006)
12. Scheider, S., Degbelo, A., Lemmens, R., van Elzakker, C., Zimmerhof, P., Kostic, N., Jones, J., Banhatti, G.: Exploratory querying of sparql endpoints in space and time. Semantic Web **8**(1), 65–86 (2017)
13. Volz, J., Bizer, C., Gaedke, M., Kobilarov, G.: Silk-a link discovery framework for the web of data. LDOW **538** (2009)