

# The YASGUI Family of SPARQL Clients<sup>1</sup>

**Editor(s):** Eero Hyvönen, Aalto University, Finland

**Solicited review(s):** Osma Suominen, Aalto University, Finland; Sébastien Ferré, University Rennes 1, France; one anonymous reviewer

Laurens Rietveld<sup>a</sup> and Rinke Hoekstra<sup>a,b</sup>

<sup>a</sup> *Department of Computer Science, VU University Amsterdam, The Netherlands*

*E-mail: {laurens.rietveld,rinke.hoekstra}@vu.nl*

<sup>b</sup> *Leibniz Center for Law, Faculty of Law, University of Amsterdam, The Netherlands*

*E-mail: hoekstra@uva.nl*

**Abstract.** The size and complexity of the Semantic Web and its technology stack makes it difficult to query. Access to Linked Data could be greatly facilitated if it were supported by a tool with a strong focus on usability. In this paper we present the YASGUI family of SPARQL clients, a continuation of the YASGUI tool introduced more than two years ago. The YASGUI family of SPARQL clients enables *publishers* to improve ease of access for their SPARQL endpoints, and gives *consumers* of Linked Data a robust, feature-rich and user friendly SPARQL editor.

We show that the YASGUI family had significant impact on the landscape of Linked Data management: YASGUI components are integrated in state-of-the-art triple-stores and Linked Data applications, and used as front-end by a large number of Linked Data publishers. Additionally, we show that the YASGUI web service – which provides access to *any* SPARQL endpoint – has a large and growing user base amongst Linked Data consumers.

**Keywords:** SPARQL, Query formulation, Data Publishing, Linked Data

## 1. Introduction

Web developers can rely on advanced development tools such as in-browser debugging, integrated development environments, high-level libraries in all but the most austere programming languages, and increasingly simple and lightweight web-services. These are essential enablers for broad take up in industry, and vice versa the use of web technology in industry drives the development of ever more developer-friendly tools. Semantic Web and Linked Data technologies have some catching up to do, and steps in this direction are under way. An example is the recent start of the W3C Linked Data Platform working group<sup>1</sup> that aims to bring triple-store querying closer to the RESTful paradigm.

Several good Linked Data programming libraries exist, but uptake of these still relies on a thorough understanding of SPARQL and the other members of the Semantic Web technology stack. The situation for SPARQL is worse. Existing SPARQL clients convey a rather narrow interpretation of what a SPARQL client interface should do: POST (or GET) a SPARQL query string to an endpoint URL. As a result, these implementations do not offer functionality that goes far beyond a simple HTML form (see section 2).

The curious developer or potential enthusiast who wants to have a first taste of Linked Data is easily scared away by the current set of SPARQL clients. For Semantic Web developers designing and testing SPARQL queries is often a cumbersome and painful experience: “All who know the RDF namespace URI by heart raise their hands now!”, and “Where is that Linked Data?”. Many will know the DBpedia endpoint URL, but can perhaps recall only a handful of endpoints in total.

---

<sup>1</sup>This work is an extended version of the YASGUI workshop paper [13]

<sup>1</sup>See <http://www.w3.org/2012/ldp>

Existing clients offer only a small selection of the features that we, as a community, could offer to both ourselves as well as new users of Semantic Web technology. We propose to overcome this hurdle by means of simple, lightweight and user friendly clients for interacting with Linked Data that integrate with existing services in the field.

This was our main motivation for designing and building Yet Another SPARQL GUI (YASGUI),<sup>2</sup> first introduced in a workshop paper [13]. YASGUI is a web-based SPARQL client that can be used to query both remote and local endpoints. It integrates linked data services and web APIs to offer features such as auto-completion and endpoint lookup. It supports query retention – query texts persist across sessions – and query ‘permalinks’, as well as syntax checking and highlighting. YASGUI is easy to deploy locally, and it is robust. Because of its dependency on third party services, we have paid extra attention to graceful degradation when these services are inaccessible or produce unintelligible results. The YASGUI family of SPARQL clients enables *publishers* to improve ease of access for their SPARQL endpoints, and gives *consumers* of Linked Data a robust, feature-rich and user friendly SPARQL editor.

In [13], we showed the added value of combining Web 2.0 and Semantic Web technologies [1,3] in this setting. Two later papers [15,14] relied on client-side query logs accumulated through YASGUI, to break Linked Data usage interpretation from the confines of single-store server side logs (see Section 4).

This paper builds on parts of [13] that illustrate the premises of this work (see below), but contains two significant new contributions. Firstly, YASGUI has grown into a family of reusable components: two new, fully client side components (YASQE and YASR) that can be used independently, and a new, more lightweight version of the full-fledged YASGUI tool. Secondly, we show that since the 2013 paper, YASGUI has had considerable impact in the field. Our components have found their way into several third-party tools, and are now incorporated in three popular triple stores. YASGUI is currently used by several important data publishers, and has shown to provide a useful information source for further research.

### Structure of the Paper

This paper is structured as follows. Section 2 is an updated version of the corresponding section in [13]

and provides an overview of the features present in the current state of the art in SPARQL user interfaces (Table 1 reflects the new situation). Section 3 builds on the description of the original YASGUI of [13] to compare and explain the features and design considerations of the new YASGUI, YASQE and YASR components. Impact of the YASGUI family is discussed in Section 4. We conclude in section 5.

## 2. State of the Art in SPARQL User Interfaces

The features of SPARQL clients can be categorized under three main headers, *syntactic* features (auto-completion, syntax highlighting and validation), *applicability* features (endpoint or platform dependent/independent) and *usability* (query retention, results rendering and download, quick evaluation). Table 1 lists seventeen SPARQL clients – that range from very basic to elaborate – and depicts what features they implement. This section describes these features in more detail, and discusses whether and how the clients of Table 1 implement these features.

We excluded query interfaces that were not fully reproducible (SPARQLinG [12], ViziQuer [19], SPARQLViz [6] and NITELIGHT [17]) or only cover a subset of the SPARQL standard (iSPARQL<sup>3</sup>)

### 2.1. Syntactic Features

Most modern applications that feature textual input support some form of *auto-completion*. Examples are the Google website which shows an auto-completion list for your search query, or your browser which (based on forms you previously filled in) shows auto-complete lists for text inputs. One advantage of auto-completion is that it saves you from writing the complete text. Another advantage is the increase in transparency, as the auto-completion suggestions may contain information the user was not aware of. The latter is particularly interesting for SPARQL, where users might not always know the exact namespace prefix they would like to use, or where the user might not know all available properties in a triple-store. Several SPARQL interfaces offer naive auto-completion functionalities, such as the Flint SPARQL Editor<sup>4</sup> which auto-completes SPARQL syntax and functions. Other

---

<sup>3</sup>See <http://dbpedia.org/isparql/>

<sup>4</sup>See <http://openuplabs.tso.co.uk/demos/sparqleditor>

---

<sup>2</sup>See <http://yasgui.org>

Feature	4Store	OpenLink Virtuoso	StarDog	ClioPatria	SNORQL	SPARQLer	Apache Jena	Sesame Workbench	Sesame2 Windows Client	TopBraid Composer	Glint	Twinkle	SparqlGUI	SparQLed	Gosparqled	Squebi	Flint SPARQL Editor	YASGUI Family
<i>Syntactic features</i>																		
Auto-completion	-	-	-	+	-	-	+	-	-	-	-	-	-	+	+	+	+	+
Syntax Highlighting	-	-	+	+	-	-	+	+	-	+	+	-	-	+	+	+	+	+
Syntax Validation	-	-	-	+	-	-	+	+	-	-	-	-	-	+	+	-	+	+
<i>Applicability features</i>																		
Multiple Endpoints	-	-	-	-	-	-	-	-	+	-	+	+	± <sup>b</sup>	-	-	± <sup>b</sup>	± <sup>b</sup>	+
Platform independent	+	+	+	+	+	+	+	+	-	+	-	+	-	+	+	+	+	+
Available as library	-	-	-	-	-	-	-	-	-	-	-	-	-	+	+	+	+	+
<i>Usability features</i>																		
Query retention	-	-	-	+	-	-	+	+	+	-	+	-	+	-	+	-	-	+
File upload	-	-	+	+	-	-	+	+	± <sup>c</sup>	+	-	+	+	-	-	-	-	- <sup>d</sup>
Results rendering	-	± <sup>e</sup>	+	+	+	± <sup>e</sup>	+	+	± <sup>e</sup>	+	± <sup>e</sup>	± <sup>e</sup>	± <sup>e</sup>	+	+	+	+	+
Chart Visualizations	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+
Results download	+	+	+	+	+	+	+	+	+	+	+	+	+	-	+	+	-	+
Endpoint Search	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	-	+

<sup>a</sup> Feature is added/improved compared to the workshop version

<sup>b</sup> Can deal with a limited number of endpoints, e.g. only CORS enabled ones.

<sup>c</sup> File upload requires a local triple store that implements the OpenRDF SAIL API, e.g. OpenRDF Sesame or Openlink Virtuoso.

<sup>d</sup> File upload is a planned feature, using cloud triple-store services (e.g. dydra.com)

<sup>e</sup> The rendering does not use hyperlinks for URI resources.

Table 1: SPARQL client feature matrix

interfaces offer more auto-completion functionalities using external APIs, such as Squebi<sup>5</sup> for prefix auto-completion, and ClioPatria<sup>6</sup> and Apache Jena<sup>7</sup> for prefix and property/class auto-completions. Editors such as SparQLed [9] and Gosparqled<sup>8</sup> offer even more reliable auto-completions of resources, though this requires a dedicated back-end server.

*Syntax highlighting* is a common functionality for programming language editors. It allows users to distinguish between different properties, variables, strings, etc. The same advantage holds for query languages such as SPARQL, where you would like to distinguish between literals, URIs, query variables, function calls, etc. The few SPARQL editors that support syntax highlighting are the Flint SPARQL Editor (or its derivatives) and Squebi, which both use the CodeMirror JavaScript library<sup>9</sup> to bring color to SPARQL queries.

Most Integrated Development Environments (IDEs) provide feedback when code contains syntax errors (i.e. runtime *syntax validation*). Feedback is immediate, which means the user can spot syntax errors in the code without having to execute them. Again, such functionality is useful for SPARQL editing as well. Immediate feedback on a SPARQL syntax means the user can spot invalid queries without having to execute it on a SPARQL endpoint. The quality of such feedback is often better compared to endpoint error messages: an IDE can pinpoint the error location in the user interface, where the returned errors from endpoints (depending on the triple-store) can differ greatly in both specificity and quality. The Flint, SparQLed, Gosparqled, Sesame Workbench, Apache Jena and ClioPatria SPARQL editors support immediate live syntax checking by means of JavaScript SPARQL parsers.

## 2.2. Applicability Features

There are only six clients that allow access to multiple endpoints. Most triple-stores provide a client interface, linking to that specific endpoint. They are *endpoint dependent*. Examples are 4Store [10], ClioPatria, StarDog<sup>10</sup>, Apache Jena, OpenLink Virtuoso<sup>11</sup>, Open-

RDF Sesame Workbench [7] and SPARQLer<sup>12</sup>. More generic endpoint *independent* clients are the Sesame2 Windows Client [7], Glint<sup>13</sup>, Twinkle<sup>14</sup> and SparqlGUI<sup>15</sup>. Other applications only provide access to *some* SPARQL endpoints. The Flint SPARQL Editor and Squebi only connect to endpoints that are CORS enabled (i.e. support cross-domain JavaScript access). This is a problem because we observe that 38% of all available endpoints<sup>16</sup> are *in-accessible* via cross-domain JavaScript. Other editors support only XML or JSON as query results, such as SNORQL<sup>17</sup> (part of D2RQ<sup>18</sup>), which only supports query results in SPARQL/JSON format.

*Platform In-dependence* increases the accessibility of a SPARQL client. The user can access the client on any operating system. Web interfaces are a good example, as a site should work on any major browser (Internet Explorer/Firefox/Chrome), and at least one of these browsers is available for any type of common operating system. Examples are the SPARQL interfaces of Virtuoso, 4Store and the Gosparqled. Another example of multi-platform support is the use of a .jar file (e.g. Twinkle), as all major operating systems support java. Examples of single-platform applications are Sesame2 Windows Client and SparqlGUI: they require Windows.

Interfaces that are open-source and *available as standalone library*, are easy to integrate into other projects and libraries. Most of the presented interfaces are either closed source, or not published as an independent library. The SPARQL interfaces that do enable such re-use, are SparQLed, Gosparqled, Squebi and the Flint SPARQL Editor.

## 2.3. Usability Features

*Query retention* enables re-use of important or often used queries, and allows users to close the application, and resume working on the query later.

*Quick evaluation* or testing of a graph generated by the user should not require the hassle of installing a local triple-store. Ideally, this functionality would be embedded in the SPARQL client application itself.

<sup>5</sup>See <https://github.com/tkurz/squebi>

<sup>6</sup>See <http://cliopatria.swi-prolog.org/>

<sup>7</sup>See <https://jena.apache.org/>

<sup>8</sup>See <https://github.com/scampi/gosparqled>

<sup>9</sup>See <http://codemirror.net/>

<sup>10</sup>See <http://stardog.com/>

<sup>11</sup>See <http://virtuoso.openlinksw.com/>

<sup>12</sup>See <http://www.sparql.org/>

<sup>13</sup>See <https://github.com/MikeJ1971/Glint>

<sup>14</sup>See <http://www.ldodds.com/projects/twinkle/>

<sup>15</sup>See <http://www.dotnetrdf.org/content.asp?pageID=SparqlGUI>

<sup>16</sup>See [sparql.es.okfn.org](http://sparql.es.okfn.org)

<sup>17</sup>See <https://github.com/kurtjx/SNORQL/>

<sup>18</sup>See <http://d2rq.org/>

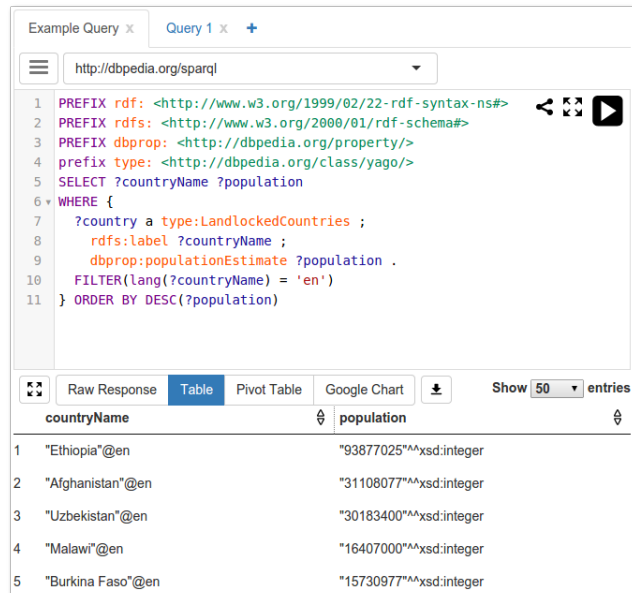
Most applications that require a local installation on the users computer support this feature, such as Twinkle. The Sesame Windows Client supports file uploads as well, though it requires a local triple-store that implements the OpenRDF SAIL API.

Query results (such as JSON or XML) for SELECT queries are often relatively difficult to read and interpret, especially for a novice. A *rendering* method which is easy to interpret and understand is a table. All applications except 4Store support the rendering of query results into a table. Because of the use of persistent URIs, we would expect navigable results for resources, e.g. in the form of drawing the URIs as hyperlinks. This feature is not supported by some applications, such as Virtuoso, Twinkle or SparqlGUI. SNORQL appears to be the application with the most elaborate way of visualizing query results: Next to a simple URI hyper-link button, the user can click on a link to browse the current endpoint for resources relevant to that URI.

Rendering the results in a tabular fashion might not suit every use case. Instead, aggregating and visualizing the SPARQL results as *charts* may be preferable. Existing Chart solution exists, such as SgVizler [16] (which indirectly uses Google Charts and D3.js), but none of the existing SPARQL editors support such drawing of charts.

*Downloading* the results as a file allows for better re-use of these results. A user might want to avoid running the same heavy query more than once, and store the results locally instead. Additionally, the results of CONSTRUCT queries are often used in other applications or triple-stores. Saving the user from needing to copy & paste query results clearly improves user experience as well. The only applications that do not support the downloading of results are the Flint SPARQL editor and SparQLed.

Most of the clients described above are restricted to one simple task: accessing information behind a SPARQL endpoint. However, equally important to this task is assisting the user in doing so. Looking at the table, the most elaborate editors are Squebi, Flint, Gosparqled and SparQLed. However, these all fall short in usability features, and most importantly: the ability to access *any* SPARQL endpoint. We conclude that currently no single endpoint independent, accessible, user-friendly SPARQL client exists.



The screenshot shows the YASGUI web interface. At the top, there is a browser address bar with the URL 'http://dbpedia.org/sparql'. Below the address bar, a SPARQL query is displayed in a code editor. The query is as follows:

```

1 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
2 PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
3 PREFIX dbprop: <http://dbpedia.org/property/>
4 prefix type: <http://dbpedia.org/class/yago/>
5 SELECT ?countryName ?population
6 WHERE {
7   ?country a type:LandLockedCountries ;
8   rdfs:label ?countryName ;
9   dbprop:populationEstimate ?population .
10  FILTER(lang(?countryName) = 'en')
11 } ORDER BY DESC(?population)

```

Below the query editor, there are several tabs for rendering the results: 'Raw Response', 'Table' (which is selected), 'Pivot Table', and 'Google Chart'. There is also a 'Show 50 entries' dropdown menu. The 'Table' view displays the following results:

countryName	population
"Ethiopia"@en	"93877025"^^xsd:integer
"Afghanistan"@en	"31108077"^^xsd:integer
"Uzbekistan"@en	"30183400"^^xsd:integer
"Malawi"@en	"16407000"^^xsd:integer
"Burkina Faso"@en	"15730977"^^xsd:integer

Fig. 1. The YASGUI interface

### 3. The YASGUI Family

The preceding section shows how current SPARQL clients fall short in supporting Linked Data access. This is both a *publisher* and *consumer* problem. From a consumer perspective, Linked Data access is difficult because available SPARQL interfaces simply do not suffice. Publishers face the problem that no SPARQL interface libraries exist that would facilitate access and lower the threshold for the potential users of their data.

This section presents the open source<sup>19</sup> YASGUI family of SPARQL clients, consisting of components targeted at both publishers and consumers. The main component is a rewritten, modularized and extended version of YASGUI, first published in [13]. YASGUI is a user-friendly web-based interface for interacting with any SPARQL endpoint. It is targeted towards consumers of linked data, and is available online at <http://yasgui.org>.

For publishers, we provide three JavaScript packages: the complete YASGUI interface, the part of YASGUI responsible for writing the SPARQL query (YASQE, or 'Yet Another SPARQL Query Editor'), and the part of YASGUI responsible for visualizing the SPARQL results (YASR, or 'Yet Another SPARQL Result-set visualizer'). To increase the ease of integration by publishers and developers, all the JavaScript li-

<sup>19</sup>See <https://github.com/YASGUI>

libraries are available via the NodeJS Package Manager (NPM), the JavaScript dependency manager Bower, and via the JsDelivr<sup>20</sup> and CDNjs<sup>21</sup> Content Delivery Networks.

Below, we first discuss the YASQE and YASR components, the used technology and services, and how the features of both JavaScript libraries compare to the tools presented in the previous section. We then present how both libraries are combined to form the YASGUI library.

### 3.1. YASQE



```

1 PREFIX foaf: <http://xmlns.com/foaf/0.1/>
2 PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
3 SELECT * WHERE {
4   ?sub foaf: ?obj .
5 }
6 LIMIT 10

```

Fig. 2. The YASQE interface

YASQE<sup>22</sup> (See Figure 2) is an extensive JavaScript library, targeted at Semantic Web publishers. YASQE takes a simple HTML text area, and – with one JavaScript command – transforms it into a full featured IDE-like SPARQL query editor.

YASQE is based on the CodeMirror JavaScript library<sup>23</sup>, an extensive HTML text editor. Using CodeMirror and the JavaScript SPARQL grammar from the Flint SPARQL Editor, YASQE is able to tokenize, highlight, validate, and dissect SPARQL queries. If needed, users are presented with immediate validation errors of their queries, and information on the type of validation error. Additionally, YASQE provides several *auto-completion* services: Full namespace URIs are completed as you type, using the Prefix.cc web service. Properties and classes are auto-completed as well, using the Linked Open Vocabularies [2] (LOV) API.

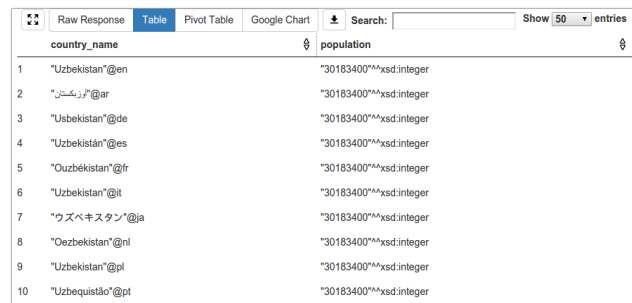
Using HTML 5 functionalities, YASQE stores that application state, making it *persistent* between user sessions: a returning user will see the screen as it was when she last closed the YASQE browser page.

Furthermore, YASQE provides query *permalink* functionality: For a given query, YASQE generates a link. Opening the link in a browser opens YASQE with the specified query filled in. We believe this is a welcome feature for people working together with a need to share queries.

Finally, YASQE has built-in support for submitting SPARQL queries to endpoints. By providing an abstract layer on top of the HTTP protocol, publishers and developers do not have to implement their own (error-prone) HTTP requests to SPARQL endpoints.

YASQE is developed to cater for many different publishing use cases, where not all features are needed all the time. To this end, YASQE is both configurable and extensible. Configurable, because publishers can toggle any of the above features on or off. And extensible, e.g. by modifying SPARQL auto-completion methods. The extensibility is concretely illustrated by the Gosparqled editor,<sup>24</sup> which takes YASQE as its main component, and adds custom auto-completion functionality.

### 3.2. YASR



	country_name	population
1	"Uzbekistan"@en	"30183400"^^xsd:integer
2	"Ўزбекистон"@ar	"30183400"^^xsd:integer
3	"Uzbekistan"@de	"30183400"^^xsd:integer
4	"Uzbekistán"@es	"30183400"^^xsd:integer
5	"Ouzbékistan"@fr	"30183400"^^xsd:integer
6	"Uzbekistan"@it	"30183400"^^xsd:integer
7	"ウズベキスタン"@ja	"30183400"^^xsd:integer
8	"Oezbekistan"@nl	"30183400"^^xsd:integer
9	"Uzbekistan"@pl	"30183400"^^xsd:integer
10	"Uzbequistão"@pt	"30183400"^^xsd:integer

Fig. 3. The YASR interface

The YASR JavaScript library<sup>25</sup> (See Figure 3), aimed at publishers as well, parses and visualizes *any* SPARQL query response.

The W3C specifies several SPARQL result formats, including XML, JSON, CSV, Turtle and RDF/XML. To decrease the load on the publisher or developer, YASR consumes any of these data formats, by parsing the results and wrapping them in an internal data representation. A first parse attempt is based on the Content-Type specified by the HTTP response. When

<sup>20</sup>See <http://www.jsdelivr.com/>

<sup>21</sup>See <https://cdnjs.com/>

<sup>22</sup>See <http://yasqe.yasgui.org>

<sup>23</sup>See <http://codemirror.net>

<sup>24</sup>See <https://github.com/scampi/gosparqled>.

<sup>25</sup>See <http://yasr.yasgui.org>

such a Content-Type header is missing or appears to be invalid, YASR tries to parse the SPARQL results on a best-effort basis.

YASR has to deal with the wide variety of possible *errors* returned by endpoints. The SPARQL protocol specifies what the endpoint request and response should look like, but leaves error handling unspecified: what HTTP error code should be sent by an endpoint, and how should error messages be communicated? As a result, triple-stores come with various ways of conveying errors. Some endpoints return the error as part of an HTML page (with the regular 200 HTTP code), or as a SPARQL query result. Others only return an HTTP error code, where only some include a reason phrase together with the error code. The latter is a best practice for RESTful services. The absence of a standard, and the failure to adhere to best practices, makes a generic robust error handling solution messy and difficult to implement. Developing such a solution requires coding and testing by trial and error. YASR decreases the publishers and developers load by wrapping such SPARQL errors in an internal data representation.

The result of the procedures described above is a JavaScript library which is capable of handling any SPARQL response, moving the burden of writing SPARQL result-set parsers and error handlers away from the publisher.

As Table 1 shows, most SPARQL clients support both *rendering* and *downloading* of query results to some extent, which YASR supports as well. Users are provided with an extensive number of visualizations: A table renderer for SELECT query responses, and another renderer for visualizing the raw highlighted query response. Next to these two simple visualizations, YASR supports visualization via Google Charts, including line, bar and scatter plots, geographical maps, and several others. YASR supports a pivotable functionality as well, allowing users to perform simple post-processing tasks on the SPARQL results. This functionality mimics functionality found in office suites such as Microsoft Excel and OpenOffice Calc, as users can cross-reference variables, aggregate on e.g. frequency counts or values, and plot these aggregated numbers on charts.

Most of the YASR visualizations are available for download, enabling offline re-use. The download options include CSV for tabular data, the as-is raw response, or the SVG renderings of charts.

Just as YASQE, YASR aims to be as extendable and configurable as possible. Publishers can easily toggle

several visualizations on and off. Thanks to the modular architecture of YASR, adding a custom visualization is easy, as developers can ignore the different SPARQL response serializations and use the internal YASR response representation directly. This is illustrated by the Visu tool<sup>26</sup>, which extends YASR by incorporating Google Chart visualizations. In turn, the Visu features have been integrated into YASR.

### 3.3. YASGUI

#### 3.3.1. JavaScript Library

The YASGUI JavaScript library<sup>27</sup> (See Figure 1) includes YASQE and YASR, adds user functionality, and wraps the libraries in a tabbed graphical user interface. Next to the features described above, YASGUI includes several usability features described below.

To increase the findability of SPARQL endpoints, YASGUI uses the SPARQLES [8] service to provide endpoint search functionality. SPARQLES is a web service which monitors the up-time and characteristics of SPARQL endpoints, in effect providing a list of *available* SPARQL endpoints. However, YASGUI only uses this information in a static fashion, as SPARQLES does not publish this information dynamically via e.g. a SPARQL endpoint or regular API. Other services and endpoint catalogs exist such as DataHub.io, but these include endpoints which are often down and unavailable, and these catalogs do not publish their data via an API accessible by JavaScript.

YASGUI also supports *user-configurable* requests. For instance, some endpoints may only support the XML results format, or allow the use of additional request parameters such as the ‘soft-limit’ of 4Store or different reasoning levels of StarDog. Such endpoints can only be used to their full potential if users are able to specify these additional arguments manually. Therefore, YASGUI supports the specification of an arbitrary number of request parameters for every endpoint.

Where YASQE and YASR make the application states persistent between browser sessions, YASGUI goes a step further. YASGUI keeps track of queries and endpoints you have accessed in the past, and allows you to restore these queries from your local history.

The features described above are all bundled in the YASGUI JavaScript library. For those publishers that require more elaborate features going beyond the possibilities of client-side JavaScript, we provide a server-

<sup>26</sup>See <https://github.com/jiemakel/visu>

<sup>27</sup>See <http://doc.yasgui.org>

side back-end as well. This light-weight back-end is written in JavaScript and runnable as a NodeJS server.

As mentioned in section 2.2, client-side web applications such as the FLINT SPARQL Editor are *endpoint independent*, but only work for endpoints that enable Cross-Origin Resource Sharing (CORS)<sup>28</sup>. To overcome this limitation, YASGUI (optionally) includes this server-side proxy to access SPARQL endpoints which are otherwise not accessible via client-side JavaScript. For endpoints which *do* support cross domain JavaScript, YASGUI executes the queries from the clients side directly.

The YASGUI server also acts as a URL shortener. Web developers deploying YASGUI can choose to use this shortener, or configure YASGUI to use one of the available web URL shorteners. The rationale behind a custom YASGUI shortener is that common web shorteners can suffer from link rot (they might disappear), they often require API key access, are not accessible from client-side JavaScript directly, and often have a limitation to the number of characters in a URL.

### 3.3.2. Web Service

Other than enabling Linked Data publishers to improve access to their SPARQL endpoints, we provide a running YASGUI instance as a web service as well<sup>29</sup>. This YASGUI instance, which includes a back-end server for CORS-disabled endpoints, presents users with a single usable editor for all SPARQL endpoints. This web service functions much like a local application: just as the regular YASGUI library, it can access SPARQL endpoints installed locally. Even more, in modern browsers, this application is still accessible when disconnected from the internet.

## 4. Impact

In our earlier work [13] we voiced our expectation that YASGUI will fill a void in the tool chain of Linked Data consumers and publishers. As in the previous sections, we tried to substantiate this expectation by giving an in depth comparison with other, similar tools, and showing that YASGUI is substantially more feature rich than the competition. Nonetheless, it was just an expectation: because YASGUI hadn't been around for very long, we could not show that this expectation rang true. This section gives a brief overview of the

impact the YASGUI family has had on the landscape of Linked Data management.

### 4.1. Integration in Triple-stores

Making YASQE and YASR available as highly configurable, lightweight, JavaScript-based front-ends for SPARQL interfaces has turned out to significantly lower the threshold for bundling YASGUI functionality with triple stores. YASQE and YASR have now made their way into three major triple stores:

#### Apache Jena

Includes both YASQE and YASR in the new Apache Jena-Fuseki 2 SPARQL interface

#### OpenRDF Sesame

Includes YASQE as its main query editor.

#### ClioPatria

Includes both YASQE and YASR as query editor

### 4.2. Integration in Other Applications

The YASGUI family reduces the effort required from other developers to program against the idiosyncrasies of SPARQL endpoints and SPARQL responses. It thereby enables developers of SPARQL applications to kick-start their user interfaces by integrating or building on top of the YASGUI tools. Until now, we have been able to find the following usage of our work in five other applications:

#### Gosparqled

An extension of YASQE, which provides (via a back-end server) smart, context-dependent auto-completions for properties and classes.

#### Visu

The first library to extend YASR with Google Chart functionality. Now published together with the YASQE editor.

#### Snapper<sup>30</sup>

An online Turtle and N-Triples editor, connecting to APIs which implement the SPARQL Graph Store Protocol. The tool uses several SPARQL queries to e.g. fetch items for auto-completions. Snapper allows users to configure such queries by means of YASQE.

#### Sefarad<sup>31</sup>

A data exploration tool-set which includes a SPARQL editor for templated SPARQL queries.

<sup>28</sup>See <http://www.w3.org/TR/cors/>

<sup>29</sup>See <http://yasgui.org>

<sup>30</sup>See <http://jiemakel.github.io/snapper>

<sup>31</sup>See <https://github.com/gsi-upm/Sefarad/>



This SPARQL editor is based on YASQE and YASR

#### **Brwsr**<sup>32</sup>

A lightweight Linked Data browser which incorporates YASQE and YASR to provide SPARQL access

### 4.3. Publishers

YASGUI components are used by a large number of publishers, in both open and closed, and non-profit and for-profit environments. Below we present a (non-exhaustive) list of Linked Data publishers that use YASGUI components. We excluded those publishers that already publish YASGUI components via their default endpoint interface, as discussed in Section 4.1.

#### **HealthData.gov**<sup>33</sup>

A US federal government website managed by the department of Health & Human Services. Access to the healthcare data is provided via YASGUI.

#### **Smithsonian**<sup>34</sup>

The Smithsonian American Art museum publishes art and artwork collections data as Linked Open Data [18]. YASGUI is used to provide access to the corresponding SPARQL endpoint.

#### **ZBW**<sup>35</sup>

The German National Library of Economics provides access to catalog information using YASQE and YASR.

#### **Linked Open Vocabularies**<sup>36</sup>

Linked Open Vocabularies is a vocabulary catalog, which publishes their data via SPARQL endpoint and via regular APIs. As discussed in section 3, YASQE uses the LOV API for auto-completion functionality. In turn, LOV uses both YASQE and YASR to provide access to their SPARQL endpoint.

#### **LOD Laundromat**<sup>37</sup>

The LOD Laundromat [4] service crawls the LOD Cloud, and re-publishes Linked Datasets

in a canonical compressed N-Triples/N-Quads format. The corresponding meta-data and provenance are stored in a SPARQL endpoint, and accessible via both YASQE and YASR.

#### **MetaLex**<sup>38</sup>

The MetaLex [11] service hosts almost all Dutch national regulations as Linked Data, and publishes these via a SPARQL endpoint. The SPARQL endpoint is accessible via the YASGUI interface.

#### **CEDAR project**<sup>39</sup>

The CEDAR project publishes Dutch census data via a SPARQL endpoint, accessible via YASQE and YASR.

#### **KennisNet**

Kennisnet is the public IT partner for educational organizations in The Netherlands. It uses YASGUI internally for accessing their SPARQL endpoint.

#### **Building Bits**<sup>40</sup>

A Semantic Web technology company which, for one of their customers, uses YASQE internally for accessing their triple-store.

#### **Kennisnet**<sup>41</sup>

Kennisnet is a Dutch institute responsible for the basic education IT infrastructure. Some of the data that Kennisnet manages and publishes are exposed via regular APIs, but internally accessible via SPARQL, using YASQE and YASR

### 4.4. Use by Consumers

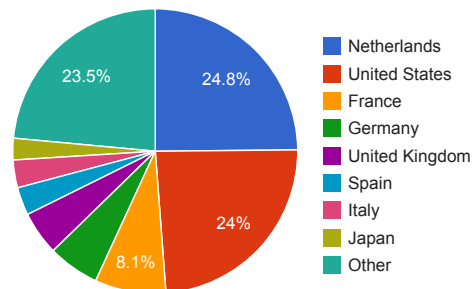


Fig. 4. Locations of YASGUI users

The YASGUI web service is publicly available since October 2012, and we have gathered usage statistics

<sup>32</sup>See <https://github.com/Data2Semantics/brwsr>

<sup>33</sup>See <http://www.healthdata.gov/sparql>

<sup>34</sup>See <http://americanart.si.edu/collections/search/ld/about/sparql.cfm>

<sup>35</sup>See <http://zbw.eu/labs/en/blog/publishing-sparql-queries-live> and <http://zbw.eu/beta/sparql-gui/>

<sup>36</sup>See <http://lov.okfn.org/dataset/lov/sparql>

<sup>37</sup>See <http://lodlaundromat.org/sparql/>

<sup>38</sup>See <http://doc.metalex.eu/query>

<sup>39</sup>See <http://lod.cedar-project.nl/cedar/data.html>

<sup>40</sup>See <http://www.buildingbits.nl/>

<sup>41</sup>See <http://www.kennisnet.nl/>

from January 2013 onwards. Over this period, we tracked (if permitted to do so) *at least* 5.200 unique visitors from over 80 countries (See figure 4), who executed 90.000 queries on around 600 endpoints.

We observe that the use of the YASGUI web service is increasing: we tracked 4.300 user sessions in 2013, which doubled to 8.400 user sessions in 2014. Note that these are conservative statistics, as only 58% of the users allowed us to track their information.

#### 4.5. Research Impact

As the USEWOD challenge [5] shows, query logs enable research in the area of the *use* of Linked Data. This challenge distributes server SPARQL query logs from 6 endpoints (including DBpedia and Bio2RDF), and has seen an impact beyond the workshop as several research papers have been published using the USEWOD query log collection.

The YASGUI service query logs contribute to this research area for two reasons. First, the YASGUI logs are solely written by real persons, allowing us to distinguish man-made queries from (routine) machine use. This is something that cannot be done using server logs alone. Secondly, the USEWOD logs cover only 6 public endpoints, while the YASGUI logs cover both open and closed Linked Data; i.e. all endpoints listed by SPARQLES as well as local, private endpoints.

Using the YASGUI logs, we were able to perform a preliminary study of the structural properties of the Linked Data cloud [15]. We also quantified the differences between server query logs as published by USEWOD and the YASGUI query logs [14], and showed how these server logs are strongly biased by machine queries, and differ greatly from those written by humans (i.e. the YASGUI logs). These studies would not have been possible without YASGUI.

To provide broader access to this unique research asset, we made the logs publicly available via the 2015 USEWOD challenge. The USEWOD log collection now has a more balanced representation of machine and man-made queries, covering the *whole* (offline and online) LOD Cloud.

## 5. Conclusion

The size and complexity of the Semantic Web make it difficult to query, and requires tools with a strong focus on usability. In this paper we presented the state of the art in SPARQL user interfaces, and showed most of

these are rather austere clients with little focus on usability, extendability, and feature completeness. Most striking is that their functionality is largely complementary: we have the SNORQL client for associative browsing, the Squebi editor for highlighted queries, several libraries which are accessible as SPARQL interface libraries, and other tools whose major selling point is access to *any* SPARQL endpoint. This large collection of tools, each with their own specific ‘area of expertise’, makes it hard for consumers to find and use the right tool for their task, and makes it time consuming for publishers to improve access to their SPARQL endpoint. Increasing user accessibility to the Semantic Web would require a tool-set which combines as much of these features as possible.

This is why we introduced the YASGUI family, target at both data *publishers* and data *consumers*. The JavaScript libraries of YASQE, YASR and YASGUI enable data publishers to easily improve access to data. The YASGUI web service allows Linked Data consumers to access *any* SPARQL endpoint – both remote and local –, and includes all the features present in the JavaScript libraries such as auto-completions, endpoint lookup, persistent user sessions, and syntax validation. In some areas there is room for improvement: we plan on extending the visualizations in YASR, and making them more intuitive to create. Additionally, we plan to improve the YASQE auto-completions using the dataset *itself*, presenting users with more suitable suggestions.

Since its launch more than two years ago, three triple-stores integrated YASGUI in their endpoint front-end, and several developers either adapted or included YASGUI components in new Linked Data applications. A large number of publishers use YASGUI components as their SPARQL endpoint interface, and close to a hundred thousand queries have been executed via the YASGUI web service on hundreds of SPARQL endpoints. The logs collected from this web service proved to be a useful data source for further research. This shows that the YASGUI family made a large impact on the landscape of Linked Data management.

## Acknowledgements

This work was supported by the Dutch national program COMMIT.

## References

- [1] Anupriya Ankolekar, Markus Krötzsch, Thanh Tran, and Denny Vrandečić. The Two Cultures: Mashing up Web 2.0 and the Semantic Web. In *WWW '07: Proceedings of the 16th international conference on World Wide Web*, pages 825–834, New York, NY, USA, 2007. ACM Press.
- [2] Thomas Baker, Pierre-Yves Vandenbussche, and Bernard Vatant. Requirements for vocabulary preservation and governance. *Library Hi Tech*, 31(4):657–668, 2013.
- [3] Robert Battle and Edward Benson. Bridging the Semantic Web and Web 2.0 with Representational State Transfer (REST). *Web Semantics: Science, Services and Agents on the World Wide Web*, 6(1):61–69, February 2008.
- [4] Wouter Beek, Laurens Rietveld, Hamid R. Bazoobandi, Jan Wielemaker, and Stefan Schlobach. LOD Laundromat: A Uniform Way of Publishing Other People’s Dirty Data. In Peter Mika, Tania Tudorache, Abraham Bernstein, Chris Welty, Craig A. Knoblock, Denny Vrandečić, Paul T. Groth, Natasha F. Noy, Krzysztof Janowicz, and Carole A. Goble, editors, *Semantic Web Conference*, volume 8796 of *Lecture Notes in Computer Science*, pages 213–228. Springer, 2014.
- [5] Bettina Berendt, Laura Hollink, Vera Hollink, Markus Luczak-Rösch, Knud Möller, and David Vallet. Usage Analysis and the Web of Data. In *ACM SIGIR Forum*, volume 45, pages 63–69. ACM, 2011.
- [6] Jethro Borsje and Hanno Embregts. Graphical Query Composition and Natural Language Processing in an RDF Visualization Interface. *Erasmus School of Economics and Business Economics, Vol. Bachelor. Erasmus University, Rotterdam*, 2006.
- [7] Jeen Broekstra, Arjohn Kampman, and Frank van Harmelen. Sesame: A Generic Architecture for Storing and Querying RDF and RDF Schema. In Ian Horrocks and James Hendler, editors, *The Semantic Web – ISWC 2002*, volume 2342 of *Lecture Notes in Computer Science*, pages 54–68. Springer Berlin Heidelberg, 2002.
- [8] Carlos Buil-Aranda, Aidan Hogan, Jürgen Umbrich, and Pierre-Yves Vandenbussche. SPARQL Web-Querying Infrastructure: Ready for Action? In Harith Alani, Lalana Kagal, Achille Fokoue, Paul Groth, Chris Biemann, Xavier Parreira, Josiane Lora Aroyo, Natasha Noy, Chris Welty, and Krzysztof Janowicz, editors, *The Semantic Web – ISWC 2013*, volume 8219 of *Lecture Notes in Computer Science*, pages 277–293. Springer Berlin Heidelberg, 2013.
- [9] Stephane Campinas, Thomas E. Perry, Diego Ceccarelli, Renaud Delbru, and Giovanni Tummarello. Introducing RDF Graph Summary with Application to Assisted SPARQL Formulation. In Abdelkader Hameurlain, A Min Tjoa, and Roland Wagner, editors, *DEXA Workshops*, pages 261–266. IEEE Computer Society, 2012.
- [10] S. Harris, N. Lamb, and N. Shadbolt. 4store: The design and implementation of a clustered rdf store. In *Proceedings of the 5th International Workshop on Scalable Semantic Web Knowledge Base Systems (SSWS2009)*, pages 94–109, Chantilly, VA, USA, 2009.
- [11] Rinke Hoekstra. The MetaLex Document Server - Legal Documents as Versioned Linked Data. In Lora Aroyo, Chris Welty, Harith Alani, Jamie Taylor, Abraham Bernstein, Lalana Kagal, Natasha Fridman Noy, and Eva Blomqvist, editors, *International Semantic Web Conference*, volume 7032 of *Lecture Notes in Computer Science*, pages 128–143. Springer, 2011.
- [12] Frederik Hogenboom, Viorel Milea, Flavius Frasincar, and Uzay Kaymak. RDF-GL: A SPARQL-Based Graphical Query Language for RDF. In Richard Chbeir, Youakim Badr, Ajith Abraham, and Aboul-Ella Hassaniien, editors, *Emergent Web Intelligence: Advanced Information Retrieval*, Advanced Information and Knowledge Processing, pages 87–116. Springer London, 2010.
- [13] Laurens Rietveld and Rinke Hoekstra. YASGUI: Not Just Another SPARQL Client. In Philipp Cimiano, Miriam Fernández, Vanessa Lopez, Stefan Schlobach, and Johanna Völker, editors, *The Semantic Web: ESWC 2013 Satellite Events*, volume 7955 of *Lecture Notes in Computer Science*, pages 78–86. Springer Berlin Heidelberg, 2013.
- [14] Laurens Rietveld and Rinke Hoekstra. Man vs. Machine: Differences in SPARQL Queries. In Bettina Berendt, Laura Hollink, Markus Luczak-Rösch, Knud Möller, and David Vallet, editors, *Proceedings of the 4th USEWOD Workshop on Usage Analysis and the Web of Data, ESWC*, Crete, Greece, 2014.
- [15] Laurens Rietveld and Rinke Hoekstra. YASGUI: Feeling the Pulse of Linked Data. In Krzysztof Janowicz, Stefan Schlobach, Patrick Lambrix, and Eero Hyvönen, editors, *EKAW*, volume 8876 of *Lecture Notes in Computer Science*, pages 441–452. Springer, 2014.
- [16] Martin G. Skjæveland. Sgvizler: A JavaScript Wrapper for Easy Visualization of SPARQL Result Sets. In Elena Simperl, Barry Norton, Dunja Mladenic, Emanuele Della Valle, Irini Fundulaki, Alexandre Passant, and Raphaël Troncy, editors, *The Semantic Web: ESWC 2012 Satellite Events*, volume 7540 of *Lecture Notes in Computer Science*, pages 361–365. Springer Berlin Heidelberg, 2015.
- [17] Paul R. Smart, Alistair Russell, Dave Braines, Yannis Kalfoglou, Jie Bao, and Nigel. Shadbolt. A Visual Approach to Semantic Query Design Using a Web-Based Graphical Query Designer. In Aldo Gangemi and Jérôme Euzenat, editors, *Knowledge Engineering: Practice and Patterns*, volume 5268 of *Lecture Notes in Computer Science*, pages 275–291. Springer Berlin Heidelberg, 2008.
- [18] Pedro A. Szekely, Craig A. Knoblock, Fengyu Yang, Xuming Zhu, Eleanor E. Fink, Rachel Allen, and Georgina Goodlander. Connecting the Smithsonian American Art Museum to the Linked Data Cloud. In Philipp Cimiano, Oscar Corcho, Valentina Presutti, Laura Hollink, and Sebastian Rudolph, editors, *ESWC*, volume 7882 of *Lecture Notes in Computer Science*, pages 593–607. Springer, 2013.
- [19] Martins Zviedris and Guntis Barzdins. ViziQuer: A Tool to Explore and Query SPARQL Endpoints. In Grigoris Antoniou, Marko Grobelnik, Elena Simperl, Bijan Parsia, Dimitris Plexousakis, Pieter De Leenheer, and Jeff Pan, editors, *The Semantic Web: Research and Applications*, volume 6644 of *Lecture Notes in Computer Science*, pages 441–445. Springer Berlin Heidelberg, 2011.