WYSIWYQ – What You See Is What You Query

Ali Khalili[1] and Albert Meroño-Peñuela[1]

Department of Computer Science, Vrije Universiteit Amsterdam, NL
{a.khalili,albert.merono}@vu.nl

Abstract. Existing Linked Data browsing user interfaces (UIs) allow users who are not familiar with Semantic Web technologies to explore interlinked datasets by generating SPARQL queries behind the scenes. This is analogous to the well-known WYSIWYG (What You See Is What You Get) paradigm, which generates the required markup in the background based on the user interactions. Nonetheless, and contrarily to the WYSIWYG approach where users are enabled to switch between the source code and the UI, there is no alternative in Linked Data browsing UIs to generate the browsing UI based on a given SPARQL query. In this paper we propose WYSIWYQ (What You See Is What You Query), a novel approach that allows people to interactively visualize a given SPARQL query as a browsing UI, which can be further enriched or re-purposed by users. WYSIWYQ aims to provide a two-way binding between SPARQL queries and RDF-based faceted browsing environments. We showcase WYSIWYQ with a proof-of-concept prototype implemented on top of the Linked Data Reactor framework and grlc APIs for SPARQL queries.

1 Introduction

Existing Linked Data browsing user interfaces (UIs) allow users who are not familiar with Semantic Web technologies to explore interlinked datasets by generating SPARQL queries behind the scenes. This is analogous to the well-known *WYSIWYG* (What You See Is What You Get) paradigm, which generates the required markup in the background based on the user interactions to hide the complexity of the underlying technology. Nonetheless, and contrarily to the WYSIWYG approach where users are enabled to switch between the source code and the UI, there is no alternative in Linked Data browsing UIs to generate the browsing UI based on a given SPARQL query. This lack of reversibility turns SPARQL queries into a passive product of the current Linked Data browsing environments. The generated queries can then only get enriched and repurposed by technical users who have the knowledge of SPARQL.

In this paper, we propose an analogous concept to the currently well-adopted WYSIWYG paradigm, called *WYSIWYQ* (What You See Is What You Query). WYSIWYQ allows users to switch between the query mode and visual mode in an interactive Linked Data faceted browsing environment.

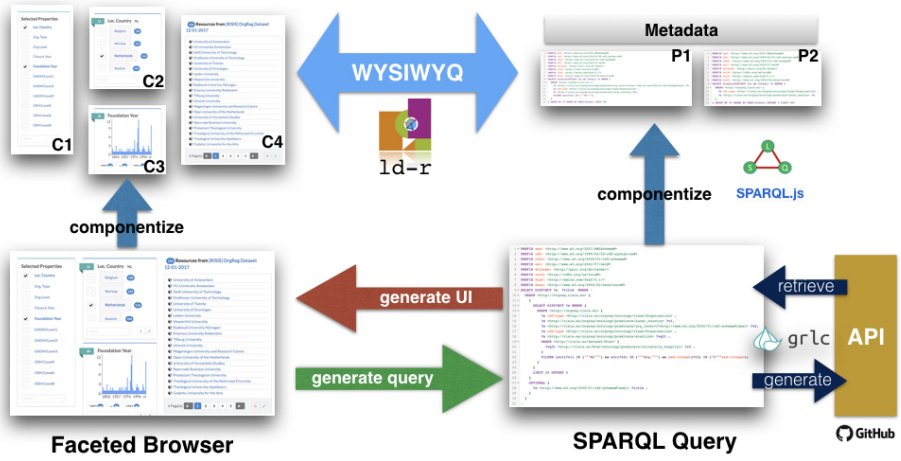

Fig. 1: The WYSIWYQ requirements together with some potential solutions.

2 WYSIWYQ Requirements

As depicted in Figure 1, our proposed WYSIWYQ model aims to enable a two-way binding between the Linked Data queries and their corresponding interactive browsing UIs, in a same way as WYSIWYG does for rich text authoring. We see the following features as main requirements for realizing the WYSIWYQ model:

- *A mechanism to componentize and customize a faceted browsing environment.* In order to be able to regenerate the faceted browsing UI, the system should provide a mechanism to decompose the monolith UI to a set of reusable and customizable components.
- *A mechanism to identify, share and enrich SPARQL queries.* In order to manage the modification of SPARQL queries in a collaborative way and to provide provenance data on changes occurred during user's interaction, the system should provide a mechanism to identify the queries as Web URIs and allow to share and modify the queries by multiple users.
- *A mechanism to validate SPARQL queries against a certain pattern.* Since it will be very hard to address all features of the SPARQL language when generating the browsing UI, the system should be able to first validate the given queries against the query patterns provided by the browsing UI and then decide whether it can render the query to an interactive UI or not.
- *A mechanism to decompose a SPARQL query into a set of sub-queries which match a certain pattern.* Given a certain query pattern, the system should be able to decompose the query into a set of fine-grained sub-queries which could be mapped into UI components in a faceted browsing environment.
- *A mechanism to map SPARQL queries and their corresponding metadata to a set of customizable UI components.* This is the core part of the system where the result of parsing a query is mapped to a set of browsing UI Com-

ponents. Based on the given metadata in the query, the system should be able to customize the generated UI components.

3 State of the Art

In the following sections we investigate the related work and available tools to fulfill the requirements identified in Section 2 and to find the existing gaps and barriers towards realizing the WYSIWYQ model.

3.1 Faceted Search and Navigation on Linked Data

The amount of Linked Data available through SPARQL endpoints is increasing[1], however, search, exploration and question answering is still a tedious task for end-users without the knowledge of SPARQL query language. There are currently two main approaches to make information retrieval from SPARQL endpoints more usable [5]: user interaction and natural language (NL). In the category of user interaction-based query generation, faceted browsing user interfaces are well-known techniques which provide a convenient and user-friendly way to navigate through a wide range of data collections [11]. Faceted browsing UIs allow users to find information without a-priori knowledge of its schema [18]. In faceted browsing the information space is partitioned using orthogonal conceptual dimensions of the data. These dimensions are called facets and represent important characteristics of the information elements. Each facet has multiple restriction values and the user selects a restriction value to constrain relevant items in the information space.

A faceted interface has several advantages over keyword search or NL queries: it allows exploration of an unknown dataset since the system suggests restriction values at each step; it is a visual interface, removing the need to write explicit queries; and it prevents dead-end queries, by only offering restriction values that do not lead to empty results [18].

There are already lots of works done in the area of faceted search and navigation over Linked Data. *SemFacet* [1] is a faceted search tool enhanced by the Semantic Web technologies to allow browsing of interlinked documents. SemFacet is implemented on top of a fragment of Yago and DBpedia abstracts. SemFacet exploits an ontology-based reasoning for generation of facets and queries. *VisiNav* [7] is another Linked Data navigation system which combines features such as keyword search, object focus, path traversal and facet selection to browse web of data with a large variance. \facet [11] is a Linked Data faceted browser that enables multi-type browsing experience and allows adapting the dynamically generated facets based on their RDF relations. It also allows users to create facet specifications to build facet dependent visualizations and interactions. *Linked Data Query Wizard* [12] is another Linked Data browsing UI, heavily

[1] 700 out of 11,238 datasets registered on `Datahub.io` provide open access to their datasets via SPARQL endpoints. [19 July 2017]

dependent on RDF Data Cube standard, which turns graph-based data into a tabular interface with supports for search and filtering to facilitate exploring linked data. *gFacet* [9] is a graph-based faceted browser which allows users to build their facets of interest on the fly. It enable users to perform a pivot operation and switch a facet to a result list. *Sparklis* [5] is a query-based faceted search UI that uses the expressivity of natural language to facilitate browsing Linked Data and understanding the generated query.

In all the above tools, SPARQL queries are treated as passive final products of the user interactions and cannot be fed back to the system as input for regenerating the faceted browsing experience. The main barrier to support this reversibility, in our opinion, is the lack of fine-grained reusable UI elements supported with a clear data-flow. Most of the current Linked Data browsers act only as a monolith Web application and thereby do not support division of UIs into a set of reusable interactive Web components to enable UI regeneration.

3.2 Componentizing Linked Data User Interfaces

Component-based software engineering (CBSE) has been an active research area since 1987 with numerous results published in the research literature [23]. Within the Semantic Web community, the main focus has been on enriching current service-oriented architectures (SOAs) with semantic formalisms and thereby providing Semantic Web services as reusable and scalable software components [24]. There have also been a few attempts to create Semantic Web Components by integrating existing Web-based components with Semantic Web technology [2,8].

When it comes to component-based development of Linked Data Applications (LDAs), the works typically fall into software application frameworks that address building scalable LDAs in a modular way. The survey conducted by [10] identified the main issues in current Semantic Web applications and suggested the provision of component-based software frameworks as a potential solution to the issues identified. The Semantic Web Framework [3] was one of the first attempts in that direction to decompose the LDA development requirements into an architecture of reusable software components. In most of the current full-stack LDA frameworks such as Callimachus[2] and LDIF[3] the focus is mainly on the backend side of LDAs and less attention is paid on how Linked Data is consumed by the end-user.

Linked Data Reactor (LD-R) framework [14] is a recent attempt to componentize LDAs into a set of adaptive and reusable ReactJS[4] Web components with a clear single-directional data flow. Its proposed component-based solution emphasizes the reusability and separation of concerns in respect to developing Linked Data applications.

[2] `http://callimachusproject.org/`
[3] `http://ldif.wbsg.de/`
[4] `https://facebook.github.io/react`

3.3 Identifying, Sharing and Collaborative Editing of SPARQL Queries

In the Semantic Web there are a number of approaches proposing some form of repository for SPARQL queries. Benchmark-oriented collections, like the Berlin SPARQL Benchmark (BSBM), the DBpedia SPARQL Benchmark [17], and SP2Bench [21] are well known examples. Saleem et al. [20] publish the Linked SPARQL Queries (LSQ) dataset, "a Linked Dataset describing the SPARQL queries issued to various public SPARQL endpoints". While the publication of these query repositories improves their findability, their reuse, repurpose, and collaborative editing is hampered by technical limitations in their access. These limitations can be partially addressed by using the W3C Linked Data Platform 1.0 specification to build HTTP gateways for "accessing, updating, creating and deleting Linked Data resources"[5], an approach followed by OpenPHACTS [6] and BASIL [4]. These systems build Linked Data APIs compliant with the Open API specification[6] that function as wrappers around SPARQL endpoints, taking responsibility of executing the SPARQL queries. This guarantees an easy, systematic, and reusable execution of SPARQL queries. However, it also harms their application coupling, since queries become part of the system's code; their re-purposing, since queries get bound to specific API operations; and their collaborative editing, since queries become non-accessible behind the API's wall. These pitfalls of traditional API-around-SPARQL approaches are addressed by grlc [16], an automatic Linked Data API builder that fetches SPARQL queries from Web accessible code publishing platforms such as GitHub. This allows for uniquely identifying, simultaneously accessing, and richly describing the provenance of both SPARQL queries and their equivalent APIs. By decoupling the API generation process from the query curation process, users can leverage the functionalities of code publishing platforms to collaboratively edit, re-purpose, fork, pull request, and document queries.

3.4 SPARQL Query Validation and Componentization

The recently proposed Shapes Constraint Language[7] (SHACL) is "a language for describing and validating RDF graphs". SHACL revolves around the so-called *shapes graph*, a graph that contains constraints that a certain target RDF graph must meet. Two important shortcomings of SHACL are that these constraints do not support named graphs, which are typically used in SPARQL queries retrieving Linked Data UI components; and that using SHACL for SPARQL validation adds yet another layer of parsing.

In our approach, we propose to use *parametrization* and *metadata* on top of SPARQL queries in order to check their compliance with UI components. These are two principles already used in grlc, and hence our proposal uses and extends grlc for the specific case of matching UI components. Listing 1.1 shows

[5] `https://www.w3.org/TR/2015/REC-ldp-20150226/`
[6] `https://github.com/OAI/OpenAPI-Specification`
[7] `https://www.w3.org/TR/shacl/`

```
#+ expand:
#+ - v1_values
#+ - v2_values
#+ browser:
#+ - BarChart: v2_values
SELECT DISTINCT ?s  ?title  WHERE {
  GRAPH <http://grid.ac/20170522> {
    {
      SELECT DISTINCT ?s WHERE {
        GRAPH <http://grid.ac/20170522> {
          ?s a ?v1.
          ?s <http://www.grid.ac/ontology/establishedYear> ?v2.
          FILTER (iri(?v1) IN (?_v1_values) && str(?v2) IN (?_v2_values))
        }
      }
      LIMIT 20 OFFSET 0
    }
    OPTIONAL {
      ?s rdfs:label ?title .
    }
  }
}
```

Listing 1.1: Example of a SPARQL query using parameters and metadata to describe a valid UI component.

an example of such a query, which uses parameters to preserve its genericity, and extends the metadata fields typically used in grlc for API generation to specify values specifically for matching and customizing UI components (e.g. to use a specific browser component to render the values of a given facet).

After a SPARQL query is validated against these patterns, tools such as SPARQL.js[8] and SPIN[9] can be employed to extract different components of the query and to semantically represent them.

3.5 Mapping SPARQL Queries to UI Components

There are already several tools available for ad-hoc and static visualization of results for a given SPARQL query. Tools such as YASGUI [19] and Sgvizler [22] enable users to benefit from different data visualizations elements (e.g. maps, charts) to render the results of a SPARQL query. However, as visualizations are the final outputs of these tools, no further interaction is supported to enrich or repurpose the given SPARQL queries and to generate a new query.

On the other hand, there are already some generic approaches to map RDF data to UI elements. WYSIWYM (What You See Is What You Mean) [13] is a generic semantics-based UI model to allow integrated visualization, exploration and authoring of structured and unstructured data. Uduvudu [15] is another approach to making an adaptive RDF-based UI engine to render Linked Data. A similar approach can be used to map elements of a SPARQL query to suitable UI elements in order to realize the WYSIWYQ model.

[8] A parser for the SPARQL: `https://github.com/RubenVerborgh/SPARQL.js`
[9] an RDF representation of the SPARQL: `http://spinrdf.org/sp.html`

4 A Proof-of-concept Implementation of WYSIWYQ

A proof-of-concept implementation of the WYSIWYQ model is available at `https://github.com/ali1k/ld-r/tree/WYSIWYQ`. We also created an screencast to showcase the concept by employing some examples: `http://wysiwyq.ld-r.org`. The implemented prototype is based on the LD-R[10] framework, SPARQL.js library and grlc query APIs stored on Github.

5 Conclusion

In this paper we presented the idea of generating interactive UI components from a given SPARQL query to facilitate switching contexts between Linked Data experts and end-users with limited knowledge of SPARQL, while browsing over Linked Data. The proposed WYSIWYQ (What You See Is What You Query) model aims to create a two-way binding between the elements of a SPARQL query and existing UI components required for an interactive faceted browsing environment. We believe that providing a continuous multi-modal user experience will spread the power of existing passive SPARQL queries to end-users who can collaboratively and interactively enrich and repurpose the queries to derive more innovative insights from Linked Data.

Acknowledgments

This work was supported by a grant from the European Union's 7th Framework Programme provided for the project RISIS (GA no. 313082).

References

1. M. Arenas, B. C. Grau, E. Kharlamov, S. Marciuska, D. Zheleznyakov, and E. Jiménez-Ruiz. Semfacet: semantic faceted search over yago. In *23rd International World Wide Web Conference, WWW '14, Seoul, Republic of Korea, April 7-11, 2014, Companion Volume*, pages 123–126, 2014.
2. M. Casey and C. Pahl. Web components and the semantic web. *Electr. Notes Theor. Comput. Sci.*, 82(5):156–163, 2003.
3. R. G. Castro, A. G. Pérez, and M. n.-G. Óscar. The Semantic Web Framework: A Component-Based Framework for the Development of Semantic Web Applications. In *DEXA '08: Proceedings of the 2008 19th International Conference on Database and Expert Systems Application*, pages 185–189, Washington, DC, USA, 2008. IEEE Computer Society.
4. E. Daga, L. Panziera, and C. Pedrinaci. A BASILar Approach for Building Web APIs on top of SPARQL Endpoints. In *SALAD2015, ISWC*, volume 1359. CEUR Workshop Proceedings, 2015. `http://ceur-ws.org/Vol-1359/`.
5. S. Ferré. Expressive and scalable query-based faceted search over sparql endpoints. In *International Semantic Web Conference*, pages 438–453. Springer, 2014.

[10] `http://ld-r.org`

6. P. Groth, A. Loizou, A. J. Gray, C. Goble, L. Harland, and S. Pettifer. API-centric Linked Data integration: The Open PHACTS Discovery Platform case study. *Web Semantics: Science, Services and Agents on the World Wide Web*, 29(0):12 – 18, 2014. Life Science and e-Science.
7. A. Harth. Visinav: A system for visual search and navigation on web data. *Web Semantics: Science, Services and Agents on the WWW*, 8(4):348–354, 2010.
8. O. Hartig, M. Kost, and J. C. Freytag. Designing component-based semantic web applications with DESWAP. In *ISWC*, 2008.
9. P. Heim, T. Ertl, and J. Ziegler. Facet graphs: Complex semantic querying made easy. *The Semantic Web: Research and Applications*, pages 288–302, 2010.
10. B. Heitmann, S. Kinsella, C. Hayes, and S. Decker. Implementing semantic web applications: reference architecture and challenges. In *5th International Workshop on Semantic Web-Enabled Software Engineering*, 2009.
11. M. Hildebrand, J. van Ossenbruggen, and L. Hardman. /facet: A browser for heterogeneous semantic web repositories. *ISWC*, pages 272–285, 2006.
12. P. Hoefler, M. Granitzer, E. E. Veas, and C. Seifert. Linked data query wizard: A novel interface for accessing sparql endpoints. In *LDOW*, 2014.
13. A. Khalili and S. Auer. Wysiwym – integrated visualization, exploration and authoring of semantically enriched un-structured content. *Semantic Web Journal*, 2014.
14. A. Khalili, A. Loizou, and F. van Harmelen. Adaptive linked data-driven web components: Building flexible and reusable semantic web interfaces. In *The Semantic Web - 13th International Conference, ESWC 2016, Heraklion, Crete, Greece, May 29 - June 2*, volume 9678 of *LNCS*, pages 677–692. Springer, 2016.
15. M. Luggen, A. Gschwend, A. Bernhard, and P. Cudre-Mauroux. Uduvudu: a graph-aware and adaptive ui engine for linked data.
16. A. Meroño-Peñuela and R. Hoekstra. Automatic Query-centric API for Routine Access to Linked Data. In *ISWC 2017*, 2017.
17. M. Morsey, J. Lehmann, S. Auer, and A.-C. Ngonga Ngomo. DBpedia SPARQL Benchmark—Performance Assessment with Real Queries on Real Data. In *ISWC 2011*, 2011.
18. E. Oren, R. Delbru, and S. Decker. Extending faceted navigation for rdf data. In *ISWC*, volume 4273, pages 559–572. Springer, 2006.
19. L. Rietveld and R. Hoekstra. Yasgui: not just another sparql client. In *Extended Semantic Web Conference*, pages 78–86. Springer, 2013.
20. M. Saleem, M. I. Ali, Q. Mehmood, A. Hogan, and A.-C. N. Ngomo. LSQ: Linked SPARQL Queries Dataset. In *ISWC*, volume 9367 of *LNCS*, pages 261–269. Springer.
21. M. Schmidt, T. Hornung, N. Küchlin, G. Lausen, and C. Pinkel. An Experimental Comparison of RDF Data Management Approaches in a SPARQL Benchmark Scenario. In *ISWC*, pages 82–97. Springer, 2008.
22. M. G. Skjæveland. Sgvizler: A javascript wrapper for easy visualization of sparql result sets. In *9th Extended Semantic Web Conference (ESWC2012)*, May 2012.
23. T. Vale, I. Crnkovic, E. S. de Almeida, P. A. da Mota Silveira Neto, Y. a Cerqueira Cavalcanti, and S. R. de Lemos Meira. Twenty-eight years of component-based software engineering. *Journal of Systems and Software*, 2015.
24. H. H. Wang, N. Gibbins, T. Payne, A. Patelli, and Y. Wang. A survey of semantic web services formalisms. *Concurrency and Computation: Practice and Experience*, 27(15):4053–4072, 2015. 10.1002cpe.3481.