To SCRY Linked Data: Extending SPARQL the Easy Way

Bas Stringer[1,4], Albert Meroño-Peñuela[2,3], Antonis Loizou[2], Sanne Abeln[1], and Jaap Heringa[1]

[1] Centre for Integrative Bioinformatics, VU University Amsterdam, NL
[2] Knowledge Representation and Reasoning Group, VU University Amsterdam, NL
[3] Data Archiving and Networked Services, KNAW, NL
[4] To whom correspondence should be adressed (`b.stringer@vu.nl`)

Abstract. Scientific communities are increasingly publishing datasets on the Web following the Linked Data principles, storing RDF graphs in triplestores and making them available for querying through SPARQL. However, solving domain-specific problems often relies on information that cannot be included in such triplestores. For example, it is virtually impossible to foresee, and precompute, all statistical tests users will want to run on these datasets, especially if data from external triplestores is involved. A straightforward solution is to query the triplestore with SPARQL and compute the required information post-hoc. However, post-hoc scripting is laborious and typically not reusable, and the computed information is not accessible within the original query. Other solutions allow this computation to happen at query time, as with SPARQL Extensible Value Testing (EVT) and Linked Data APIs. However, such approaches can be difficult to apply, due to limited interoperability and poor extensibility. In this paper we present SCRY, the ***S**PARQL **c**ompatible se**r**vice la**y**er*, which is a lightweight SPARQL endpoint that interprets parts of basic graph patterns as calls to user defined services. SCRY allows users to incorporate algorithms of arbitrary complexity within standards-compliant SPARQL queries, and to use the generated outputs directly within these same queries. Unlike traditional SPARQL endpoints, the RDF graph against which SCRY resolves its queries is generated at query time, by executing services encoded in the basic graph patterns. SCRY's federation-oriented design allows for easy integration with existing SPARQL endpoints, effectively extending their functionality in a decoupled, tool independent way and allowing the power of Semantic Web technology to be more easily applied to domain-specific problems.

1 Introduction

The Semantic Web continues to grow, reaching an ever growing number of scientific communities [11]. This is driven in part by the adoption of Linked Data principles and convergent practices by these communities [8], which publish a great variety of linked scientific datasets in the Linked Open Data (LOD) cloud.

This cloud currently contains over 600K RDF dumps (37B triples), ready to be queried through 640 SPARQL endpoints[3].

The diversity of available Linked Data is matched by the diversity of its consumers and their needs. For example, statisticians may want to exclude outliers from their analysis, or filter results based on the p-value of some statistical test; geographers typically need to select coordinates which fall within a certain area or distance from another point; and bioinformaticians often use shared evolutionary ancestry to transfer information between entities.

These and many other cases rely on information which is either impossible or impractical to materialize in triplestores beforehand. Whether or not an observation should be treated as an outlier depends on how one defines outliers, and the observations it is being compared with. One could precompute all pairwise distances between coordinates, but this scales quadratically with the number of entries and precludes queries spanning multiple datasets. Bioinformaticians use many different methods to predict evolutionary relatedness between biomolecules, and interpretting their results is highly context-dependent. More generally, solving domain-specific problems typically requires domain-specific tools and algorithms, whose outputs can not always be sensibly precomputed. Thus, querying such information requires it to be derived at query time.

Several approaches enabling the generation of new data and relations at query time already exist:

- The SPARQL query language includes built-in functions for basic arithmetic and string handling, and widely supported extensions are available for the most common forms of data processing, such as datatype-aware handling of literals annotated with XML schema [5]. However, such general extensions can not facilitate the diverse set of domain-specific algorithms and procedures required by many users.
- SPARQL 1.1 [7] allows the definition of customizable procedures attached to a specific URI via Extensible Value Testing (EVT), which is currently supported by several triplestore vendors. However, EVT has some fundamental limitations: custom procedures are restricted to appear in limited query environments (e.g. `BIND()`, `FILTER()`), and queries incorporating them are not interoperable between endpoints.
- Linked Data APIs[1,6] offer access to Linked Data in Web standard formats [14] without requiring users to have extensive knowledge of RDF or SPARQL. They offer access to custom procedures through user-friendly interfaces, accessing Linked Data under the hood. Such APIs enable functional extension of Linked Data queries in a more flexible way than EVT, but greatly restrict interoperability with other Linked Data sources and the type of information that can be retrieved.
- Several SPARQL endpoints allow expert users to define custom functions under the hood, e.g. Virtuoso, Jena and Stardog. Although very powerful, these features typically have a steep learning curve and, like EVT, are not interoperable with other endpoints.

Each of these approaches varies in terms of flexibility, interoperability, ease of implementation and user-friendliness. We argue many scientific communities would benefit from a combination of SPARQL's flexible, efficient manner of querying RDF data, and user-friendly access to easily customized procedures which generate RDF data at query time.

In this paper we present SCRY, the ***S**PARQL **c**ompatible se**r**vice la**y**er*. SCRY is a lightweight SPARQL endpoint that allows users to define their own services, assign them to a URI, and incorporate them in standards-compliant SPARQL queries. These services take RDF data as input and return RDF data as output, allowing users to generate and incorporate relevant information at query time. SCRY leverages SPARQL's query federation protocol to maintain interoperability with other SPARQL endpoints. Essentially, this embeds API-like functionality into pure SPARQL queries, in a standards-compliant format.

2 Problem Definition

Domain-specific questions often require domain-specific solutions, particularly with regard to information and relations which must be derived at query time because they are impractical to precompute. Currently available approaches facilitating this are limited in terms of flexibility, interoperability, user-friendliness, ease of implementation, or a combination thereof. We propose to address these issues by executing services at query time, generating requested data on demand. Consider the following query:

```
SELECT * WHERE { ?array stats:mean ?mean ;
                        stats:sd   ?sd   . }
```

If treated as a standard graph pattern, this query would only return arrays which have their mean and standard deviation materialized in the triplestore. However, if interpretted as service calls, the query engine could execute matching statistical procedures for `stats:mean` and `stats:sd`, and return bindings with a mean and standard deviation generated at query time.

Derived values like means and standard deviations are impractical to materialize statically in a dataset, but there are many scenarios where making them query-accessible is useful, if not essential, to answer domain-specific questions. Thus, the problem we address in this paper is to access Linked Data in a manner which (1) combines SPARQL's flexibility and efficiency with the functional extension provided by Linked Data APIs or under-the-hood endpoint customisation, (2) coexists and integrates with extant SPARQL tools and endpoints by complying with current standards and (3) is easy for users to extend with their own domain-specific services.

3 SCRY

Our **S**PARQL **c**ompatible se**r**vice la**y**er (SCRY) acts as a lightweight SPARQL endpoint, granting users access to easily customized services during query execution. SCRY allows services and their inputs to be encoded by URIs in the basic graph patterns of SPARQL queries. Users must configure an instance of SCRY, which we will hereafer refer to as an *orb*, with a set of services and associated URIs. Whenever a SCRY orb is queried, it searches for these URIs in the query's graph patterns and executes the associated services, prior to resolving the query itself. Upon execution, services generate RDF data, populating an RDF graph against which the original query will be resolved. Thus, what sets SCRY apart from traditional endpoints, is that it resolves queries against RDF data generated at query time, rather than against a persistent RDF graph.

Services accessed through SCRY can involve simple tasks like rounding off a number, or running complex secondary programs using local or remote resources. Typical use involves sending a query to any conventional SPARQL endpoint, which then invokes SCRY through a federated query. Information retrieved from the primary endpoint's persistent RDF graph can be used as input for a service made available through a personalized, locally hosted SCRY orb. The SCRY orb then generates an RDF graph by executing the encoded services, evaluates the federated query against said graph, and returns the results to the primary endpoint (see Figure 1). Use case-driven examples are given below.

This federation-oriented design is completely compliant with current standards, allowing SCRY to be used with any federation-capable primary endpoint. However, it also means SCRY necessarily inherits a susceptibility to network latency, from the way in which the SPARQL protocol implements query federation. Using HTTP to push serialized SPARQL queries and RDF data back and forth is relatively expensive in terms of overhead, which is particularly wasteful if the computational steps to get from input to output are short and straightforward.

SCRY is implemented in Python, using the RDFLib package [9] to interpret and resolve SPARQL queries, and the Flask microframework [13] to handle federation via HTTP. Services must thus be accessible from Python, either by being implemented as Python code or via calls to the shell, e.g. with `os.system()`. SCRY's source code, including the services demonstrated below, is available at `https://github.com/bas-stringer/scry/`.

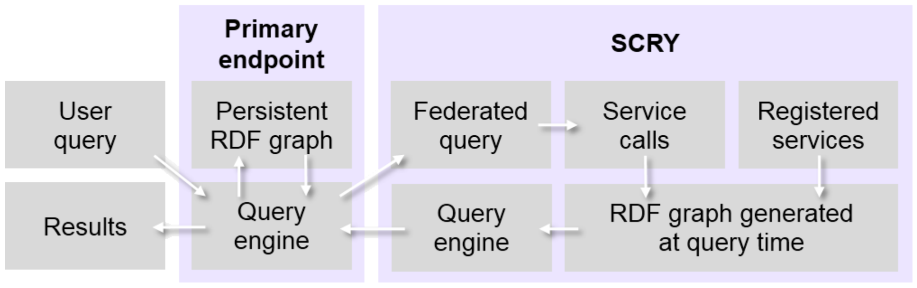

Fig. 1. Dataflow diagram of a typical SPARQL query using SCRY, through federated queries from a primary endpoint.

Table 1. Lines of code needed to extend various SPARQL endpoints with support for statistical functions. Besides being shorter, extended SCRY orbs are compatible with any SPARQL compliant endpoint by design, avoiding the need to rewrite similar extensions for different endpoint implementations.

Function	**SCRY**	**Jena**	**Virtuoso/SQL**	**Virtuoso/C**	**Stardog**
Std. deviation	2	13	10	12	89
Pearson's r	2	19	27	33	91

3.1 Use Case 1: Statistics

We have implemented several services to supplement SPARQL's basic built-in arithmetic, for example to calculate the standard deviation of an array, and the Pearson correlation between two arrays[5]. In SCRY, these can be implemented in as little as 2 lines of code each – roughly an order of magnitude fewer lines than needed in Jena, Virtuoso or Stardog (see table 1).

Social historians running the CEDAR project published statistical data of Dutch historical censuses [12]. They can now run queries which include, for example, the standard deviation of the population counts of 1899 [6].

Likewise, the Linked Statistical Data Analysis project[7] provides Linked Data for various metrics, including several precomputed statistics such as Kendall's τ correlation. However, we are interested in Pearson's r correlation instead. Querying the raw data through their endpoint and federating it to a SCRY orb allows us to calculate Pearson's correlation coefficient between, for example, *infant mortality rate* and *corruption perception indices* in 2009 [8].

3.2 Use Case 2: Bioinformatics

Homology is one of the most important concepts in bioinformatics. It is a term used to indicate two entities share evolutionary ancestry, which suggests those entities have a similar biological function. Thus, knowledge of an entity can cautiously be inferred from knowledge about its homologs.

The Bio2RDF project has compiled one of the largest collections of biological Linked Data, comprising nearly 12B triples which describe 1.1B unique entities [4]. More recently published sources of RDF data, such as neXtProt [10] and the Human Protein Atlas (HPA) [15], are not yet included therein.

Given the sheer volume of biological RDF data, making homology a query-accessible property would have many applications in bioinformatics. To this end, we have implemented a procedure that runs the BLAST program[2]: the most commonly used method to find homologs, cited nearly 55 000 times to date[9].

[5]See `http://bit.ly/stats-impl`

[6]See query at `http://bit.ly/scry-sd`

[7]See `http://stats.270a.info/.html`

[8]See `http://bit.ly/transparency-270a`

[9]Citations counted by Google Scholar.

Table 2. Results of an integrative query using SCRY's BLAST procedure to find the number of tissue-specific co-expressed homologs of hemoglobin β. The first column shows the name of tissues in which hemoglobin β is expressed. The second column shows the number of its homologs coexpressed in that tissue.

?tissue	**?N**
Alveolar macrophage	3
Bone marrow hematopoietic cell	4
Gall bladder glandular cell	3
Hemangioblast	4
Hepatic stellate cell	4
Hepatocyte	4
Lung macrophage	3
Pneumocyte	3

The Human Protein Atlas lists which proteins are found where in the human body. This information is exposed as RDF, which we have loaded in a private primary endpoint. From this endpoint, we can now federate queries to a SCRY orb to invoke services. Using our BLAST procedure, for example, we can investigate coexpression: for a given query protein, we ask the primary endpoint in which tissues it is expressed; we invoke the BLAST service through a federated query to find the protein's homologs; and we ask the primary endpoint how many of those homologs are expressed in the same tissues - within a single SPARQL query. Running such a query for hemoglobin β reveals it is found in 8 different tissues, and that at least 3 of its homologs are found in each of those tissues (see table 2).

4 Conclusions

An ever increasing number of scientific communities are adopting Semantic Web technology and Linked Data principles. Their many domain-specific problems require equally many domain-specific solutions. This is especially true when considering derived information, which is impractical to precompute, and thus must be generated at query time.

We present SCRY, an easily customized, lightweight SPARQL endpoint that facilitates executing user-defined services at query time, making their results accessible immediately within SPARQL queries. Custom procedures are implemented with relative ease, whether they perform simple statistical analysis or run complex secondary programs like BLAST. We find that extending SPARQL in this novel way is (i) an order of magnitude faster than extending other SPARQL endpoints, and (ii) compatible with any existing SPARQL 1.1 compliant endpoint.

These benefits come at the cost of a dependence on SPARQL's implementation of query federation. In particular, network latency can become an issue. Despite this limitation, SCRY provides a platform through which statistics, bioinformatics, and a variety ofz other scientific disciplines can incorporate domain-specific programs and algorithms within SPARQL queries, better enabling these diverse communities to harness the power of Semantic Web technologies.

Many roads are open for the future. First and foremost, we intend to develop a community-managed service repository, through which users can share and receive feedback on the services they implement. Furthermore, we plan on extending this work by implementing: (i) a browser-based query interface, allowing users to query their SCRY orb directly (i.e. not through federated queries); (ii) efficiency, security and authorization features, which will make it feasible to host public SCRY orbs; and (iii) more domain-specific procedures, to further demonstrate SCRY's versatility and enable more scientific communities to harness the power of semantic web technologies.

Acknowledgements. The authors wish to express great gratitude towards Frank van Harmelen and Paul Groth, for their advice and feedback during the project; resident Python guru Maurits Dijkstra for his support with development and implementation of the program; and Laurens Rietveld and Ali Khalili for their valuable comments on this manuscript.

References

1. Linked Data API. Tech. rep., UK Government Linked Data (2009), `https://github.com/UKGovLD/linked-data-api`
2. Altschul, S.F., et al.: Gapped BLAST and PSI-BLAST: a new generation of protein database search programs. Nucleic Acids Research (1997)
3. Beek, W., et al.: LOD Laundromat: A Uniform Way of Publishing Other People's Dirty Data. In: ISWC 2014 (2014)
4. Belleau, F., otherse: Bio2RDF: Towards a mashup to build bioinformatics knowledge systems. Journal of Biomedical Informatics (2008)
5. David C. Fallside, P.W.: XML Schema Part 0: Primer Second Edition. Tech. rep., World Wide Web Consortium (2004), `http://www.w3.org/TR/xmlschema-0/`
6. Groth, P., et al.: 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), `http://www.sciencedirect.com/science/article/pii/S1570826814000195`, life Science and e-Science
7. Harris, S., Seaborne, A.: SPARQL 1.1 Query Language. Tech. rep., World Wide Web Consortium (2013), `http://www.w3.org/TR/sparql11-query/`
8. Heath, T., Bizer, C.: Linked Data: Evolving the Web into a Global Data Space, vol. 1:1. Morgan and Claypool, 1st edn. (2011)
9. Krech, D., et al.: RDFLib Python Library. Tech. rep. (2002), `https://github.com/RDFLib/rdflib`
10. Lane, L., et al.: neXtProt: a knowledge platform for human proteins. Nucleic Acids Research (2011)
11. Max Schmachtenberg, Christian Bizer, A.J., Cyganiak, R.: Linking Open Data cloud diagram 2014. `http://lod-cloud.net/` (2014)
12. Meroño-Peñuela, A., Guéret, C., Ashkpour, A., Schlobach, S.: CEDAR: The Dutch Historical Censuses as Linked Open Data. Semantic Web – Interoperability, Usability, Applicability (2015), under review
13. Ronacher, A., et al.: Flask Python micro web application framework. Tech. rep. (2010), `http://flask.pocoo.org/`
14. Sporny, M., et al.: JSON-LD: A JSON-based Serialization for Linked Data. Tech. rep., World Wide Web Consortium (2014), `http://www.w3.org/TR/json-ld/`
15. Uhlé, M., et al.: Tissue-based map of the human proteome. Science (2015)