An open-source frontend for RDF-based knowledge graphs

Tobias Malmheimer and Magnus Pfeffer
{malmsheimer,pfeffer}@hdm-stuttgart.de

Introduction

Knowledge graphs representing concepts and their semantic relationships have become a digital humanities mainstay. They consolidate domain knowledge in a single data structure that enables a new spectrum of data-driven research opportunities [1]. Our ongoing research project strives to create such a knowledge graph for the domain of Japanese visual media, especially manga, anime, and computer games, by cooperating with fan communities on the web to re-use their curated data [2].

Data is collected from multiple sources and converted into the RDF format. One of the core characteristics of this format is that all entities and attributes are represented as URIs, while the value of said attributes are either URIs (thus linking two entities using a property) or literal values. This allows for an iterative approach to integrating and merging the information into the final knowledge graph, while preserving provenance information on the statement level [3].

There are various RDF triple store software packages that can be used to store and access the data and the SPARQL language can then be used to formulate search queries on RDF stored in such a database, but this requires the user to be both familiar with the query language as well as the structure of the RDF data.

As all entities and properties are identified by URIs, one way to explore RDF data is having a web server that serves the domain that the data URIs are residing in and shows all information that can be associated with a given URI. This functionality is one of the main ideas of linked data: a linked data frontend can serve “raw” RDF data to programs that try to resolve an URI while human users using a browser to resolve the same URI get a human-readable HTML view of all the data that is associated with this URI. Such a frontend also allows for easy exploration and navigation of a dataset, as all URIs in the human-readable view can be made into clickable links. See Figure 1 for an example.

![Figure 1: Human-readable HTML view of all the information associated with an URI that is representing a character from a computer game](image-url)
Most RDF triple stores, especially commercial solutions, come with a simple web frontend that provides the exploration capabilities described above. But these often come with a limited set of configuration options. As an open-source alternative, there is the “Pubby” frontend, which was developed in the D2ME side-project of the Europeana initiative [4]. This software is quite versatile and includes functionality such as a configurable SPARQL-query and SPARQL-endpoint, content negotiation, external label lookup, a preferred language for labels and preloading labels for faster response. But there are several trade-offs: It is developed as a Java Web Application and needs the corresponding infrastructure to be run as a server. Also there is a significant slowdown when a URI entity page has a lot of incoming or outgoing links, as the labels for these links are resolved using individual lookups.

A universal frontend

As the JVMG knowledge graph has entity pages with as much as 90,000 labelled links where the performance becomes a big issue and we desire more control over the appearance and further functionality of the web frontend, a new universal web frontend for RDF-based knowledge graphs is being developed as part of the research project. It is implemented using the Python programming language and the Django web application framework. In order to have fast label lookups, our approach creates a single SPARQL query that retrieves all relevant data for a given URI and the corresponding labels. This minimizes the amount of connections to the database and allows the database to use its internal structures (e.g. indexes and caches) to speed up query processing. Figure 2 shows a simplified version of our SPARQL query. For a given URI (Target) it gathers all triples where the URI is either the subject or the object. Additionally, it also gathers the labels of each part of a triple.

![Fig. 2: Simplified version of our SPARQL query](image)

Python code then generates a basic HTML view that can be further styled using CSS. It is a rather small codebase that – besides the Django framework – mainly relies on the SPARQLWrapper and RDFlib modules. The response times for entity pages with many links are several orders of magnitude faster than the prior “Pubby” installation.

Key functionality

As the code base is very compact and easy to understand, extended functionality can easily be added. Correspondingly, the HTML view has evolved quite a bit and now includes

- multiple CSS variants, including a “dark mode”
- provenance information on every statement to indicate the data source
- settings to limit the labels to one or more languages
- interactively expandable attribute sections that limit the number of values by default to keep the view compact
- filters to hide information contained in specific subsets of the data (separate graphs in the triple store)
The framework allows for the development of plug-in-like expansions that add new functionality to the server. Guided by the application of the knowledge graph data in small research use cases by media science researchers, some expansions were developed that showcase the possibilities:

- **Co-occurrence counter**
  This expansion uses the currently shown URI in the web frontend as a starting point and traverses the graph to collect all entities that are linked to it and all their literal and URI attribute values. These values are then tallied and result in a co-occurrence statistic for the starting entities. This is completely independent from the starting point: starting from a tag describing a character trait, the list will show other traits and many characters share a given combination. Starting from a person associated with a work, the list will show other persons that have worked together with the starting person and the number of works this collaboration encompasses.

- **Elasticsearch**
  A simple search interface has been added that uses an Elasticsearch index. One can search for label literal values or parts of URI strings

**Future work**
One aspect that needs further work is performance. Very large requests like the aforementioned character entity page no longer crash or time out, but response times are still too long. The current bottleneck seems to be the database and we are looking into the configuration options of the Fuseki triple store we currently use. Also, we are developing more examples for plug-ins that extend the basic navigation functionality in a meaningful way. Currently being tested are prototypes for preparing data to be passed to visualisation software, exporting parts of the knowledge graph in formats other than RDF and others. Documentation is still very limited and the code has changed a lot in a short time. The prototype frontend is running very stable and has already been adopted in another research project, so we feel comfortable to release the code to the public soon.

**References:**


