RDF Navigation Tool Documentation

Nicolas Lasolle
nicolas.lasolle@univ-lorraine.fr

Université de Lorraine, CNRS, Université de Strasbourg,
AHP-PReST, F-54000 Nancy, France

Université de Lorraine, CNRS, Inria, LORIA, F-54000 Nancy,
France

May 2021

Introduction

This document intends to describe the installation, the configuration and the use of a navigational tool for the exploration of Semantic Web databases. It has been designed to find interesting resources by exploiting similarities between elements of an RDF graph. A resource can be anything such as an art work, a document, a place, an individual, etc. The results found using the tools are exposed through a chronological-based interface in which one can navigate through results. First designed for the Henri Poincaré correspondence corpus, this system can be reused with any corpora as long as corpus data is available through a SPARQL endpoint. This document is divided into three chapters: an installation and configuration guide, a presentation of the user interface and an explanation of the system architecture and the source code organization.

Contents

1 Installation and configuration guide **3**
- 1.1 System requirements . . . 3
  - 1.1.1 Java . . . 3
  - 1.1.2 Web browser . . . 3
  - 1.1.3 SPARQL endpoint . . . 4
- 1.2 Installation . . . 4
  - 1.2.1 GitHub repository . . . 4
  - 1.2.2 Installation process . . . 4
- 1.3 Configuration . . . 5

2 Presentation of the user interface **8**
- 2.1 Overview of the Tool . . . 8
- 2.2 Add new filtering conditions . . . 12

3 System architecture **14**
- 3.1 Developer requirements . . . 14
- 3.2 Global architecture . . . 14
- 3.3 Backend . . . 14
  - 3.3.1 Maven configuration with the pom.xml file . . . 14
  - 3.3.2 API deployment . . . 15
  - 3.3.3 Source code architecture . . . 16

Chapter 1

Installation and configuration guide

The system is decomposed into two distinct applications: a Web user interface defined using the HTML, CSS and Javascript technologies which manages the user interaction as well as the visualization and the export of results, and a Java application which interact with the RDF database by executing generated SPARQL queries.

1.1 System requirements

1.1.1 Java

The Java application used to communicate with the SPARQL endpoint requires the installation of Java on the machine. The recommended Java version for this application is Java 8 or above, which can be downloaded here (`https://www.oracle.com/java/technologies/javase-jre8-downloads.html`). Installing a JRE version should be enough to make the application functional. Please pay attention to your system configuration (i.e. the operating system and your platform architecture). If you want to have a look at the source code and need to make some modifications, it is recommended to download a JDK instead, which is precised again in chapter 3.

1.1.2 Web browser

The application user interface is a Web interface which should be used through a Web browser. No specific browser is recommended for this application, it should be working with any of them. During the development process, the Mozilla Firefox browser has been used. The up-to-date version of this browser can be downloaded here (`https://www.mozilla.org/fr/firefox/new/`).

1.1.3 SPARQL endpoint

For using this tool, it is necessary to make a connection with an RDF database through a SPARQL endpoint. SPARQL (SPARQL Protocol and RDF Query Language) refers to a data transmission protocol and query language which can be used to access and to update RDF data. A SPARQL endpoint is a environment dedicated to the execution of SPARQL queries over an RDF database. You can find many tutorials for exposing your RDF data through a SPARQL No more details are provided here as it is not the purpose of this document.

The default configuration file creates a connection to the DBpedia SPARQL endpoint (`http://dbpedia.org/sparql`). DBpedia is a community effort to extract and to structure content from information created in various Wikimedia projects. The Wikimedia movement aims at providing educational content to the world through diverse initiatives and projects whose best known is Wikipedia. For the default use case, the objective is to retrieve some information related to literary works.

1.2 Installation

1.2.1 GitHub repository

The system is accessible online on a GitHub repository (`https://github.com/nlasolle/rdf_navigation_tool`). On this repository, on the `app` folder, one can find a Java application, packaged as a launchable Jar file named `rdf_navigation_tool-X.X.X.jar`, and a Web application which is composed of several source files (.html, .css and .js) located in the `app/web` folder. Both parts are required to make the application functional. The application relies on the use of configuration file: several configuration files are provided in the `app/config` folder. In the `sources` folder, you can retrieve the full Java application source code which is not required for this installation—see chapter 3) for technical details about the tool.

1.2.2 Installation process

1. Download the content of the `app` folder from the GitHub repository.
2. Extract this folder into the desired repository on your machine.
3. Open a terminal at the `app` folder location
4. On a Linux environment, it may be necessary to make the jar file exutable using the following command:

```
sudo chmod +x ./rdf_navigation_tool-1.0.0.jar
```

5. Execute the following command:

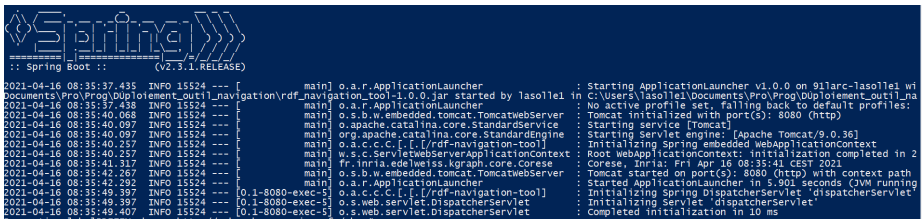

Figure 1.1: *Terminal content after application launch.*

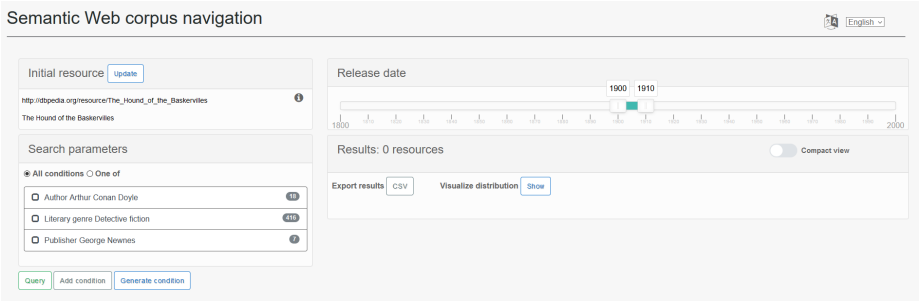

Figure 1.2: *The navigation tool interface.*

```
java -jar ./rdf_navigation_tool-1.0.0.jar --spring.config.name=./config/application_dbpedia_1
```

6. This should launch the application using the given configuration file related to a DBpedia use case. The result should be similar to the terminal content presented in Figure 1.1.

7. Open the `index.html` file, located in the `app/web`, with any Web browser.

8. The Web page content should be similar to the interface presented in Figure 1.2.

If you encountered a problem during this test process, please check the version of Java installed on your system. This command gives the JRE version which is installed on your system:

```
java -version
```

1.3 Configuration

When developing this tool, one of the main challenges was to ensure its reusability with other corpora. In this context, the Java application comes with a specific configuration file which can be used to define several values. This file is defined

using the YAML (*YAML Ain't Markup Language*) format which is a human friendly data serialization standard for several programming languages.

The configuration file is composed of two kinds of parameters:

server parameters related to the API configuration.

custom custom parameters that should be updated when creating an application use case for a corpus exploration. Several configuration files may be created for the same corpus if it is required to explore different kind of resources or by using other similarity measures.

The description of server parameters is given on Table 1.1. The description of custom parameters is given on Table 1.2. If you need to validate and format your YAML configuration file, you can use this online tool.[1] If you want to better understand the YAML language, please find the following tutorial[2] which provides a lot of useful information. Some parameters values are optional, but in this situation it is necessary to let the parameter name with an empty value rather than removing the whole parameter line.

Table 1.1: Server properties with their descriptions.

Name	Mandatory	Description
port	No	The port for exposing the REST API which allows the communication with the Web interface (set to 8080 by default).
address	No	Network address to which the server should bind.
sessionTimeout	No	Session timeout in minutes (default is 30 minutes)
servlet.context-path	Yes	The path for API methods access. If you change this value, you should update the API calls in the frontend application source code otherwise the Web application will fail to reach the API.

[1] `http://www.yamllint.com/`.

[2] `https://www.cloudbees.com/blog/yaml-tutorial-everything-you-need-get-started/`.

Table 1.2: Custom properties with their descriptions.

Name	Mandatory	Description
initialResource	Yes	The IRI of the initial resource for corpus exploration
sparqlEndpoint	Yes	The URL of a public SPARQL endpoint
itemLink	No	A base URL for accessing resources representations. If not provided, the tool will open a tab using the resource IRI when clicking on the resource. As an example, the DBpedia resources IRI redirects on HTML representations of resources.
transformationRulesFile	Yes	The path to the transformation rule file used to generate new filtering conditions.
prefixes	Yes	A comma separated list of RDF prefixes with the associated namespaces. Please ensure that you have included all prefixes used in the configuration file.
pickedProperties	Yes	A comma separated list of RDF properties used to generate filtering conditions related to the initial resource.
displayedProperties	Yes	A comma separated list of RDF properties to be displayed for each result
dateProperty	Yes	The RDF property which is used for chronological-based results filtering and presentation
dateOptions	Yes	The filtering options for the date slider
labels.individual	Yes	The label property which should be used to display textual representation of graph resources
labels.property	Yes	The label property which should be used to display textual representation of ontology properties
labels.language	Yes	The language tag (e.g. 'fr', 'en', etc.) to be used when retrieving any textual representation

Chapter 2

Presentation of the user interface

This chapter presents the user interface of this system which can be used for the exploration of any Semantic Web graph. It is particularly relevant for the exploration of historical corpora because results are presented within a chronological-based interface. A demonstration video of the use of this tool for the Henri Poincaré correspondence corpus is available online.[1]

This section introduces the tool through a use case related to the search of literary works. Some details about the works are retrieved by querying the DBpedia public SPARQL endpoint. This use case corresponds to the configuration file named `application_dbpedia_1.yml`. Other DBpedia use cases are provided on the `config` folder: `application_dbpedia_2.yml` is related to music albums search by starting a process with the 1975 Herbie Hancock's album named *Man-Child*; `application_dbpedia_3.yml` is related to paintings search by starting a process with the 1937 Pablo Picasso famous work named *Guernica*.

2.1 Overview of the Tool

The tool is available as a Web interface which allows users to generate SPARQL queries and to visualize, to filter and to export results. An excerpt of the user interface in use is presented in Figure 2.3. It is is divided into four main blocks.

Initial resource block

The top-left block gives some information about the initial resource related to the current search process. In our example, this resource corresponds to the literary work *The Hound of the Baskervilles* written by Arthur Conan Doyle in 1902. The block presents the IRI associated to the resource as well as its label. Clicking on one of these elements redirects on a human-readable representation

[1] `https://videos.ahp-numerique.fr/videos/watch/f90ff003-39db-4b4c-ade6-fb18b86d9244`.

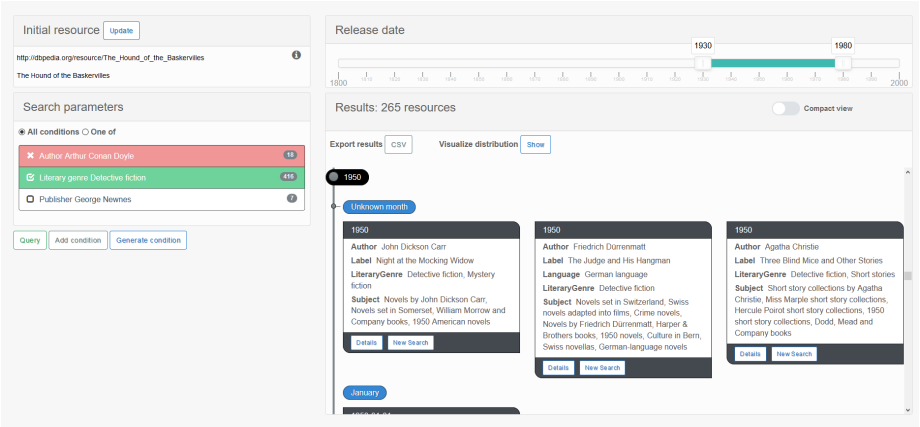

Figure 2.1: *The navigation tool interface in use for the exploration of the DBpedia graph.*

of the resource. In our example, this redirects to an HTML page[2] giving all available data about the *The Hound of the Baskervilles* description on DBpedia. The info box on the top-right corner provides a description of the resource based on the `rdfs:comment` property value.

The "update" button opens an advanced search modal window in which one can retrieve a resource from the RDF graph with a search based on its type and its label. Some resources may be missing from that search window because of query limitations imposed by the SPARQL endpoint.[3] The interface lets users give the IRI of the resource if it is not presented in the list. Once the selection of a resource has been validated, a new application process is initialized centered around this resource. As an example, if one search for the label "Jekyll", he/she will be proposed the famous novella *The Strange Case of Dr. Jekyll and Mr. Hyde* from Robert Louis Stevenson. If this resource is selected and one clicks on the "validate" button, a new process is launch and the result would be similar to the one presented in figure. In this situation, the proposed filtering conditions are different from the ones related to the *The Hound of the Baskervilles* (which is the default resource used for the example).

The bottom-left block gathers a set of search conditions which can be used to create SPARQL queries. These conditions are generated based on the initial resource and are presented in a human-readable form by reusing the labels associated with the ontology properties and with the graph resources. For the running example, this leads to the generation of three conditions related to the author, the genre and the publisher of the literary work. The properties that must be used by the tool to generate conditions are specified in a configuration file. Other meta-data exist related to literary works, some of them could

[2] `https://dbpedia.org/page/The_Hound_of_the_Baskervilles`.

[3] As an example, DBpedia endpoint limits query results to 10,000 rows.

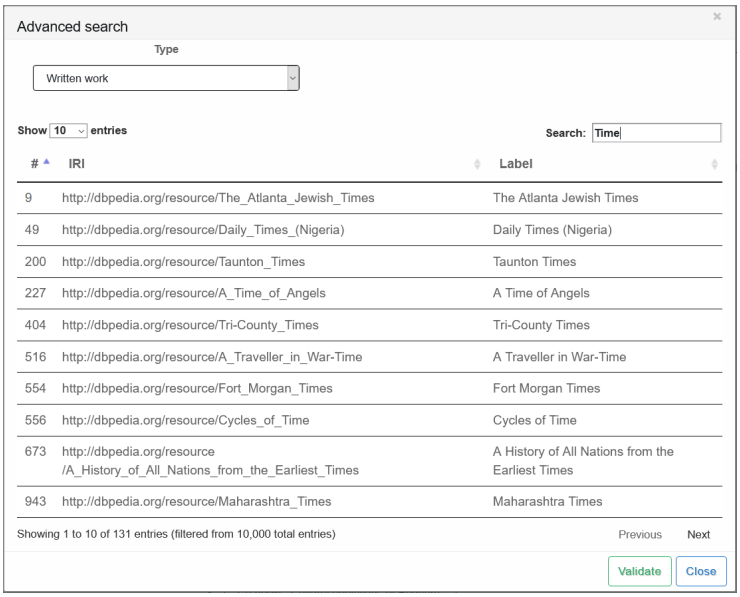

Figure 2.2: *The advanced search window used to find a resource and start a new application process.*

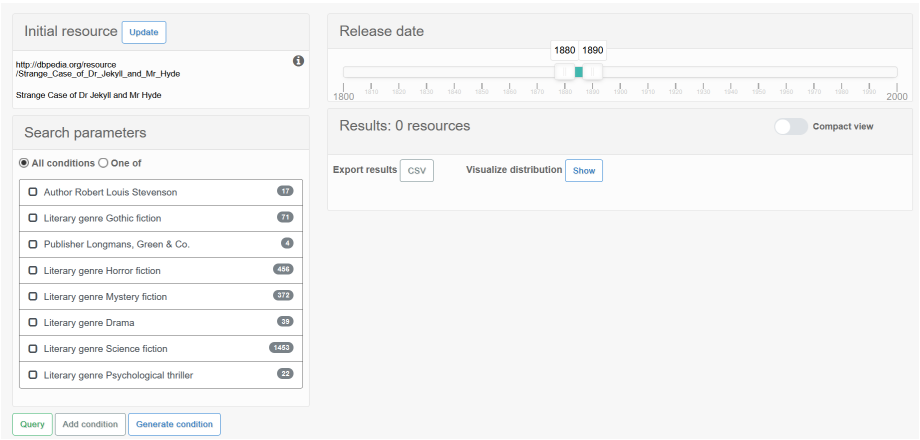

Figure 2.3: *The interface content after starting a new process related to* The Strange Case of Dr. Jekyll and Mr. Hyde.

be interesting for As an example, the property which specifies the number of pages of the original document has not been identified as relevant and is not considered by the tool when generating conditions, though this property could easily be added by editing the configuration file. Next to each condition is given a number which corresponds to the number of resources matching the given condition on the RDF graph. Users can select different conditions by clicking on them. A first click makes the condition box background green and means that the system considers resources matching the condition. A second click makes the background red and means that the system considers resources which do not match this condition. Another click puts the condition in its initial state, meaning that it is not considered by the system for the search over the RDF graph. Users can choose to look for resources matching all conditions ("All conditions" mode) or at least one of them ("One of" mode).

Clicking on the "query" button updates the results presented on the bottom-right block of the interface. For the running example, the conditions lead to the formulation of the following query.

$$\mathcal{Q}_{search} = \left| \begin{array}{l} \text{Give me the works associated with the } \texttt{detective} \\ \texttt{fiction} \text{ literary genre which have not been written} \\ \text{by } \texttt{Arthur Conan Doyle}. \end{array} \right.$$

The result block presents a set of resources in a chronological-based view. For each result, some information about the resource is given. For this use-case, each resource is described with a label, the authors, the literary genres, and the subjects which provides details about the topics and the context of works. These properties are configurable and can be different from the properties used to generate conditions. Clicking on the resource box redirects on a human-readable representation of the resource. The system proposes to export the results in a CSV file which embeds the IRI and some information about each resource. Another functionality concerns the presentation of a bar chart which expresses the distribution of the results in relation to the chosen date property. In the figure 2.4 is presented the distribution of works associated with the detective fiction literary genre. Distribution data can be exported to a CSV file.

It is also possible to start a new search process centered around one of the resources presented in the result block. The idea of the system is to start with an initial resource and to explore the corpus by navigating from a resource to another by taking different paths. This way of exploring the corpus could lead to the identification of unexpected relations between the elements of the corpus.

Above the result block, a date slider can be used to filter the set of presented results. This filtering tool is related to a configurable date property. In our situation, this property corresponds to the literary work release date and it can be set between 1800 and 2000. The initial range of the slider is centered around the date related to the selected resource.

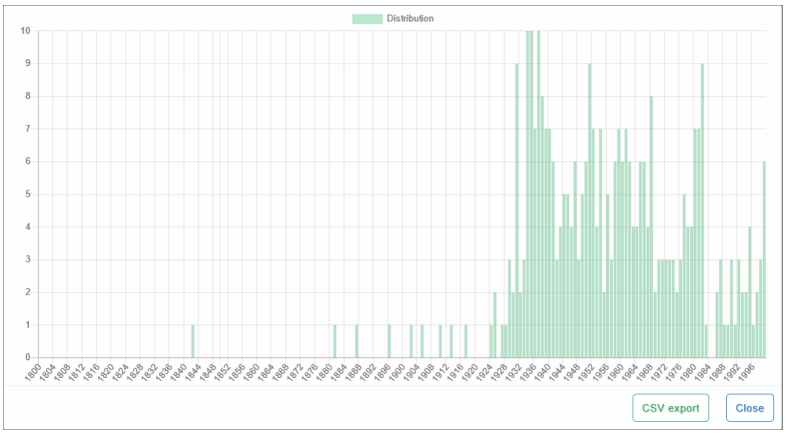

Figure 2.4: *The distribution of works associated with the detective fiction literary genre.*

2.2 Add new filtering conditions

In some situations, the generated conditions may not be sufficient for providing interesting or surprising results. A first solution, which is included in the tool, is to let users manually add a new condition. For this purpose, the "Add manual condition" button shows a new window in which users can create conditions by selecting one of the available properties and by attributing a value that can be retrieved from a list or manually entered if necessary. As an example, Figure 2.5 corresponds to the addition of a new condition related to the *location city* property. This functionality may be useful in several situations but it can be a long process to find and select the appropriate properties and values. Moreover, adding a condition which is not at all related to the initial resource would not have a lot of sense in a navigation tool that relies on resource similarities.

Another idea to overcome this issue is to be able to generate new conditions that the initial resource would not necessary match but which are related to its characteristics. An implementation of this idea, based on the use of transformation rules have been set for the Henri Poincaré correspondence corpus but requires some adjustments for using it with other corpora. It should be available in the next application release.

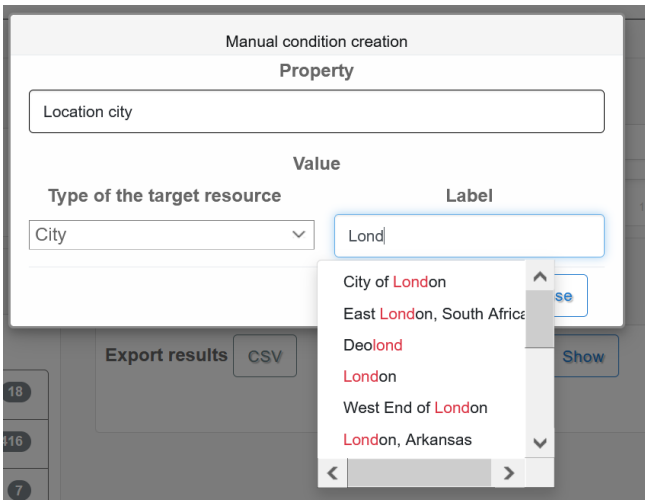

Figure 2.5: *Window used to manually add filtering conditions.*

Chapter 3

System architecture

3.1 Developer requirements

Java

To run the application in development mode, please install a JDK (version 8 or above) which can be downloaded here (`https://www.oracle.com/java/technologies/javase/javase-jdk8-downloads.html`).

Maven

Apache Maven is a build tool which has been used for the management of dependencies, unit tests and the archive creation. The last stable version is available on the Maven website.[1] Maven relies on the creation of a configuration file, named pom.xml, which contains information about the project and configuration details used by Maven to build the project.

3.2 Global architecture

3.3 Backend

3.3.1 Maven configuration with the pom.xml file

Project archetype

A first set of properties allows to describe the project:

```
<groupId>org.ahp</groupId>
<artifactId>rdf_navigation_tool</artifactId>
<version>1.0.0</version>
<packaging>jar</packaging>
```

[1] `https://maven.apache.org/download.cgi`

Parent project

A second set of properties is related to the parent project which is defined to enable our project to inherit some objects from the parent project. In this situation, the Spring Boot framework[2] is used to create the application. Spring boot allows the creation of stand-alone Spring applications. It is a common Java solution for creating REST API as it enables a lot of easy to use objects to facilitate the initialization of the project.

Dependencies

Another set of properties that is defined in this configuration file lists the project dependencies. Here are the list of defined dependencies and their use:

- Spring boot: application architecture management and creation of the REST API;
- Swagger: API documentation;
- JUnit: writing and execution of unit tests;
- Jena: Semantic Web library for executing SPARQL queries over the endpoint;
- SQTRL: transformation rule engine use to generate alternative filtering conditions;
- javatuples: creation of simple tuple object;
- jackson-dataformat-yaml: for managing the update of the YAML configuration file.

Plugins

A last set of properties contains a list of plugins.

3.3.2 API deployment

The API can be deployed by executing the following command from a terminal open at the project root:

```
mvn package -DskipTests
```

An executable jar is created in the `target` folder. You can then launch this jar with or without specifying a configuration file, as explained in the configuration chapter. The API documentation is available at `http://localhost:8080/rdf-navigation-tool/swagger-ui.html`. This documentation is generated thanks to the SWAGGER plugin, which relies on different annotations and comments defined to describe application source code components.

[2] `https://spring.io/projects/spring-boot`.

3.3.3 Source code architecture

To be updated.

Acknowledgements

This work is supported partly by the French PIA project "Lorraine Université d'Excellence", reference ANR-15-IDEX-04-LUE.