Open Web Ontobud: An Open Source RDF4J Frontend

Francisco José Moreira Oliveira
University of Minho, Braga, Portugal
a78416@alunos.uminho.pt

José Carlos Ramalho
Department of Informatics, University of Minho, Braga, Portugal
jcr@di.uminho.pt

Abstract

Nowadays, we deal with increasing volumes of data. A few years ago, data was isolated, which did not allow communication or sharing between datasets. We live in a world where everything is connected, and our data mimics this. Data model focus changed from a square structure like the relational model to a model centered on the relations. Knowledge graphs are the new paradigm to represent and manage this new kind of information structure.

Along with this new paradigm, a new kind of database emerged to support the new needs, graph databases! Although there is an increasing interest in this field, only a few native solutions are available. Most of these are commercial, and the ones that are open source have poor interfaces, and for that, they are a little distant from end-users.

In this article, we introduce Ontobud, and discuss its design and development. A Web application that intends to improve the interface for one of the most interesting frameworks in this area: RDF4J. RDF4J is a Java framework to deal with RDF triples storage and management.

Open Web Ontobud is an open source RDF4J web frontend, created to reduce the gap between end users and the RDF4J backend. We have created a web interface that enables users with a basic knowledge of OWL and SPARQL to explore ontologies and extract information from them.

2012 ACM Subject Classification Information systems → Web Ontology Language (OWL); Information systems → Ontologies; Computing methodologies → Ontology engineering; Information systems → Graph-based database models

Keywords and phrases RDF4J, Frontend, Open Source, Ontology, REST API, RDF, SPARQL, Graph Databases

Digital Object Identifier 10.4230/OASIcs.SLATE.2020.15

Category Short Paper

1 Introduction

An ontology [32] comprises a formal naming, representation and definition of the categories, properties and relations between the concepts, data and entities/individuals that substantiate one or many domains. Ontologies are very easy to visualize as a graph, where nodes represent concepts/entities, and edges represent relations between the concepts. Using these formalities, knowledge can be defined in a simple way. Because of this, computers can use rules and logic, like the *syllogisms* to extract additional knowledge using inference[13].

RDF4J [24] is an open source Java framework developed to manipulate and/or access RDF data. This framework enables the storage of RDF based ontologies (RDF [17], RDFS [5], and OWL [1]) and their exploitation and management with SPARQL [27] (W3C language and protocol for querying web ontologies). RDF4J works both locally and remotely thanks to its REST API and implements a SPARQL endpoint for all the ontologies stored in it. RDF4J fully supports SPARQL1.1 [28], all mainstream RDF file formats, and RDFS inference.

SPARQL is a query language designed to explore RDF ontologies, and SPARQL 1.1 is an improvement over the original by adding update queries, capable of manipulating an ontology. Inference [14] uses a reasoner and a set of rules to generate new relations between resource in the ontology. These relations are often known as implicit triples. RDFS inference mechanism looks only at RDFS vocabulary within the ontology when generating new relations.

These basic features make RDF4J a versatile and capable tool, which many databases such as GraphDB, Amazon Neptune and Virtuoso have used as their foundation and then improved with additional features [24].

Despite being often used as a stepping stone framework, RDF4J remains a useful RDF Store, still receiving updates and new versions, and its Server and Workbench can be used by anyone thanks to its Open Source license [25, 8]. However the updates focus heavily on bug fixes and sometimes a few new features for the RDF4J Server, while the RDF4J Workbench remains untouched [26].

Open Web Ontobud is a web frontend that brings all the RDF4J features to the user in a simple and complete interface, creating an appealing and easy to use workbench. With its simplicity any user with a basic RDF/OWL and SPARQL knowledge is able to analyze, extract and manipulate information from ontologies.

This section introduced ontologies, RDF4J, its qualities and shortcomings, and the motivation to develop Ontobud. Section 2 presents currently existing frameworks, discusses and compares their strengths and flaws. Section 3 shows the current state of Ontobud and its components, explains its design choices and each component purpose. In Section 4 the paper is concluded and future work is presented.

2 Existing Ontology Frameworks and Databases

Before developing a new web frontend, we analyzed the already existing options. Firstly, some RDF Stores do not have a workbench or frontend, and others provide only a SPARQL query editor. Secondly, we would prefer an open source solution.

The next subsections will talk about some of the existing options that provide an interface and list what features they lack and why they were not chosen.

2.1 GraphDB

GraphDB is a highly efficient, robust, and scalable graph database [10], owned by Ontotext. GraphDB is built on top of the RDF4J framework, using it for storage and querying, and has ample support for query languages, such as SPARQL and SeRQL, and RDF syntax's, like Turtle, RDF/XML, N3, and many others [10]. The usage of all the support provided by RDF4J makes GraphDB easy to use and compatible with industry standards.

It adds the capacity to execute OWL reasoning [12] and is one of the few RDF databases that can perform semantic inferencing at scale, handling heavy query loads and real time inferencing [11].

To support different user demands, GraphDB comes with three commercial editions: GraphDB Free, GraphDB Standard Edition and GraphDB Enterprise Edition [11]. Unfortunately, the commercial nature of GraphDB goes against our desire for an open source solution and prevents us from selecting it as a valid option. Out of all the options provided, only the Free Edition would be viable, but it is still not entirely open source. GraphDB has a good interface, targeting the maintenance and exploitation of stored ontologies, but its main focus is the speed optimizations, allowing fast operations even for massive volumes of data. On the other side, there is a lack of user-centered options. For example, SPARQL queries can only be stored globally, which is not the best way since they are often ontology dependent.

2.2 Blazegraph

Blazegraph is an open source [3], ultra-high-performance graph database used in commercial and government applications [2, 4].

Blazegraph supports different operating modes (triples, provenance, and quads), and RDF/SPARQL API, allowing its use as a RDF Store, and covers application needs from small to large applications, up to 50GB statements stored in NanoSparqlServer [4].

Despite its incredible performance and adaptability, Blazegraph workbench provides only basic tools, such as executing a SPARQL Query and Update, exploring a resource by URI, and listing namespaces. Blazegraph focus is its database features, speed, and size, resulting in a poor workbench, making its management harder. This is a noticeable problem as Wikidata [33, 34], one of Blazegraph biggest users, does not use the default Blazegraph workbench and instead uses the CodeMirror [6] framework to build its workbench, which offers more features such as premade query templates, history, among others.

2.3 Neo4j

Neo4j is an open source, NoSQL native graph database, and comes with a Community and Enterprise edition [20]. In terms of speed, Neo4j delivers constant time traversals for both depth and breadth thanks to its efficient representation of nodes and relationships, all while maintaining a flexible property graph schema that can be adapted and changed at any time [20].

To explore stored ontologies, Neo4J uses Cypher [20]. Cypher is a declarative query language similar to SQL but optimized for graphs, currently used by other databases such as SAP HANA Graph and Redis graph via the openCypher project. However, when compared with SPARQL, it is much more low level. Additionally, up to version 3 (version 4 was recently released), Neo4J only allowed the storage of one single ontology. Fortunately, version 4 fixed this limitation.

Unfortunately, Neo4J natively does not have any inference engine and does not support SPARQL. There are some attempts to work around this problem using plugins such as GraphScale [19], which shows promising results but still has limitations and is currently working on improvements [16].

3 Development of Open Web Ontobud

After analyzing the currently available options and their features, we concluded that no one fulfilled our requirements. This conclusion gave the motivation and led us to propose and develop the Open Web Ontobud, an Open Source RDF4J Frontend.

Ontobud is not just a simple frontend, being divided into four main components: Frontend, Backend, MongoDB, and RDF4J Server, as can be seen in Figure 1. Each of these components, which will be discussed in detail in the following subsections, can be deployed using Docker, or if Docker is not available, a Dockerless deployment is also possible.

The frontend, as the main component, is intended to provide the interface the user will use to interact with and to manipulate his ontologies. The backend exists, most importantly, to manage the authentication process for the users and their access control, but also adds some new queries and facilitates the communication between frontend and databases.

RDF4J was the chosen RDF Store for this project. It can be any RDF4J Server, but a version 3.x or higher is advised, as a few Ontobud functionalities are not supported by earlier RDF4J versions. MongoDB stores all information that RDF4J cannot handle, such as user account information, preferences, context, and saved queries.

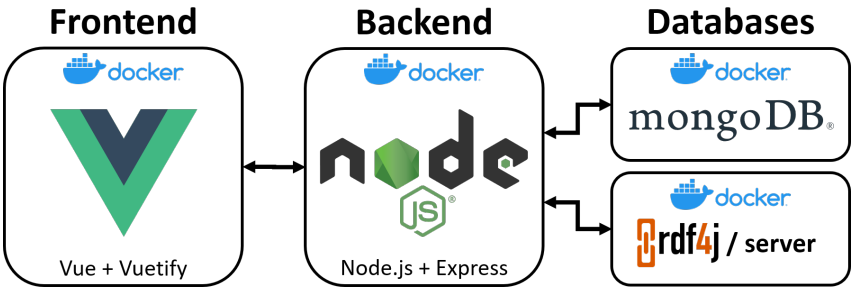


Figure 1 Ontobud Architecture.

3.1 Frontend

The frontend goal is to have an intuitive user interface capable of providing some statistical information for any given ontology, and easy access to frequently queried data. This way, new users to RDF/OWL and SPARQL can use this readily available information while learning the workings and the SPARQL queries behind it. Furthermore, experienced users can benefit from easy access to information by running fewer SPARQL queries.

Vue [30] and Vuetify [31] are the frameworks used to develop Ontobud frontend. Vue is a JavaScript Framework for the development of reactive web frontends. Vuetify is a JavaScript and CSS framework developed for Vue that adds new components with many possible configurations.

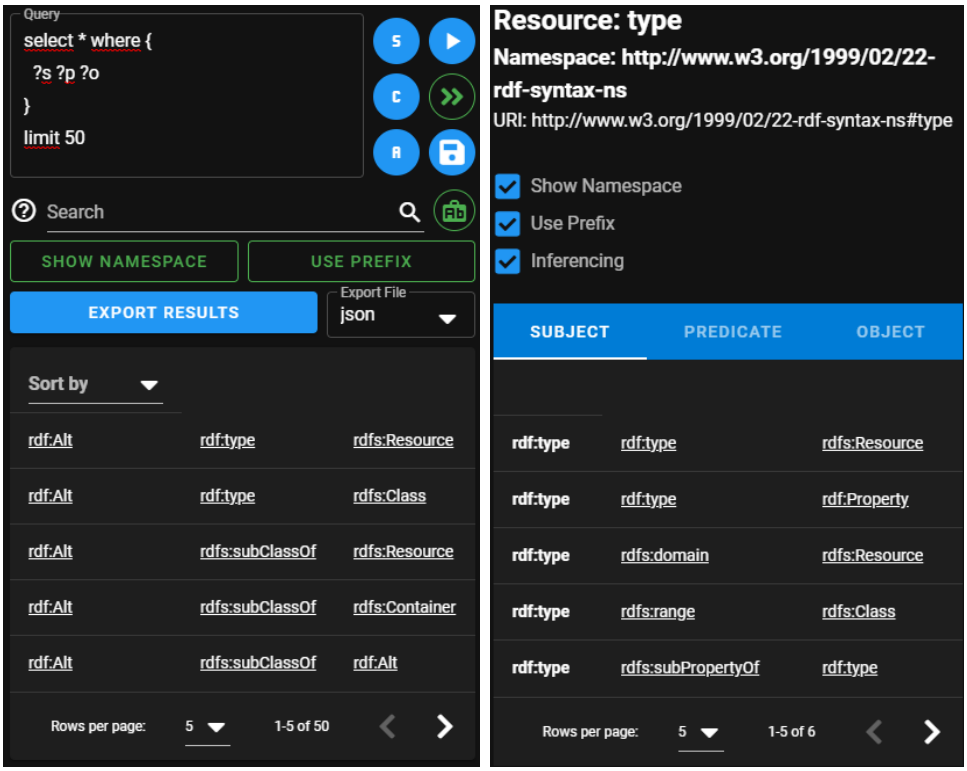
Currently, the frontend allows many operations, most notably:

- Account creation, allowing access to saved queries;
- Repository list and management (add and delete);
- Repository information such as explicit and implicit statement number, used namespaces and a list of existing classes;
- Import and export repository (accepts multiple RDF syntaxes);
- SPARQL 1.1 Query and Update execution, including:
  - SPARQL Query syntax verification (using PEG.js [23] and following the SPARQL Grammar [29]);
  - Query results search, export, and navigation;
- Saved queries:
  - repository specific (not global) or usable in all repositories (global);
  - management (user can add, reuse, edit and delete).

Figure 2a shows the results of a query placed in a table and Figure 2b shows the navigation table, where we can see all triples related to a specific URI. All URI are links that can be clicked and appear underlined to distinguish from literals, which cannot be clicked. This allows the user to navigate the ontology from any query result or resource.

By using a web browser as an interface, which is commonly pre-installed on most computers nowadays, Ontobud Frontend comes with no pre-requisites allowing any person to use the frontend. This enables anyone to use this platform anywhere, including cellphones and tablets, as the responsive nature of Vue allows the page to fit the screen with small changes to the original code.

The frontend is only a component in this project, and due to its open source nature, anyone can modify the original design, improve it, or fix bugs. It would be possible to create alternate interface designs, such as a design oriented to users unfamiliar with SPARQL, or a



(a) Query results. **(b)** Navigation.

Figure 2 Query results and navigation.

one-page design if the users want. The room for customization and improvement is there. With multiple interface designs, each user could run their favorite frontend design, made by themselves or someone else, in their computer while connected to a remote backend.

Running the frontend on your computer can be done using Docker [7] or NPM [21], both simple options with few requirements.

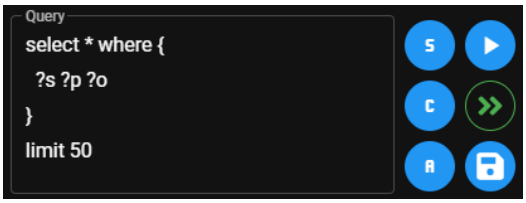
3.1.1 SPARQL syntax parser using PEG.js

Using PEG.js [23] online, we defined a parser to detect SPARQL Query syntax errors. For this, we followed the SPARQL Grammar [29] as a reference. SPARQL Update is currently not included in this parser, and the frontend will warn the user about it, as shown in Figure 3c.

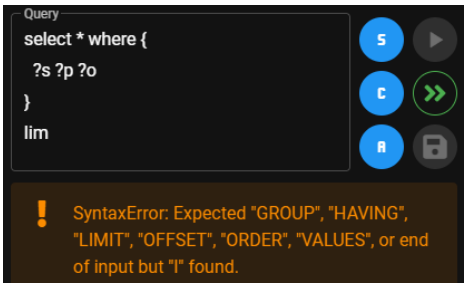
We then downloaded the parser from PEG.js and integrated it into the frontend. This allowed us to notify the user about errors in real-time, as the warning is updated whenever the query changes, and not when executed. This can be seen in Figure 3b. Figure 3c shows a warning, notifying the user that we cannot verify SPARQL Update queries, namely inserts and deletes, and Figure 3a shows a correct SPARQL Query, and therefore no warning is displayed.

3.2 Backend

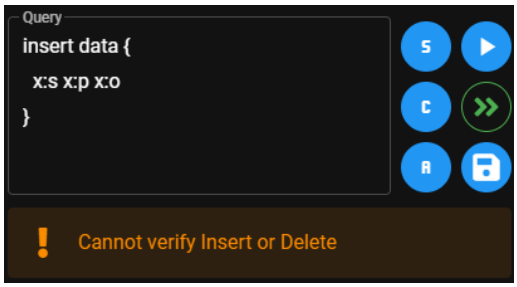
Access control and user authentication are the main focus of the backend. The frameworks used are Node.js [21], and Express [9]. User accounts require an email (account ID), password, and account name. The created account will be processed, having its password encrypted using the NPM [21] package *bcrypt* [22], and then be saved in MongoDB.



(a) SPARQL Query without errors.



(b) SPARQL Query with error.



(c) SPARQL Update notification.

Figure 3 SPARQL Query editor.

User authentication requires the account email and password. Upon authentication success, the backend will return a JSON Web Token [15] (JWT), which will be used for access control, created using asymmetric encryption. The returned JWT will allow the backend to verify if an authenticated user has permission to execute its request.

At this time, the authentication is implemented, and the user access control is currently in development. The access control will mainly allow admins access to special features, such as non-admin account management, and no restrictions in their actions. It also enables request verification from non-admins, for example, preventing userA from deleting an userB saved query.

Furthermore, the backend has its own REST API, divided into authentication, MongoDB, and RDF4J routes. The first two manage the user accounts and access to MongoDB information, respectively. The RDF4J grants access to the repositories and its routes were inspired by the original REST API, but are not all identical. We did this in an attempt to create a more friendly and simple API for users unfamiliar with the original one or interacting with API in general. Currently, not all routes provided by RDF4J are mapped into the backend, only the most common and necessary for the frontend, but direct access to the RDF4J Server is possible if needed.

The frontend uses most of the existing API to supply its functionalities but some of the provided backend routes remain unused, in particular:

- Account information – Returns a JSON object containing user information, such as, account name, email and other information fields to be added;
- Delete user account – Allow an user to delete its account or an admin to delete any account;
- Repository configuration information – Return information about a repository, such as, repository name and ID, which storage type is using and if it is using inference;
- Repository contexts – Returns a list context identifiers from a repository;

3.3 Databases

For data persistence two databases were chosen. The ontologies are stored in RDF4J [24] and the remaining information is stored in MongoDB [18]. Most notably, Ontobud uses the REST API to access the SPARQL endpoints, allowing all its components to run locally or in different machines.

MongoDB is a NoSQL document database, storing data in JSON-like documents [18]. MongoDB is used to store all information RDF4J cannot handle. This includes information such as user account, preferences, information, context and saved queries. User accounts exist to enable access control while saved queries assist the user in its work. We decided to save queries on the server-side because this allows the user access to them on any computer. Creating an account is fast, meaning new users can quickly start saving queries.

4 Conclusion

Along the paper we discussed the development of a frontend for RDF4J. The intent behind this project, which led us to propose and implement Ontobud, was to create an open source and easy to use platform. We compared the possible ontology frameworks and selected the one fitting our requirements.

The application introduced in this paper aims to improve the user experience, for both new and experienced users in the ontology domain, by providing an intuitive interface without cutting on functionalities and an easy access to analytic information about the ontology. We will test it in the context of an academic class with students recently introduced to ontologies.

4.1 Future work

As future work, we aim to finish the user access control, improving the safety in a multi-user environment, and implement a graph visualization and navigation component for a more natural exploration and understanding of any given ontology. The current navigation system presents information in tables by returning triples where a given resource URI is defined as a subject, predicate or object, but this can be confusing and overwhelming to new users.

Additionally, we plan to improve the inference mechanism of Ontobud. RDF4J can make RDFS inferences on uploaded ontologies but lacks OWL inference. This issue is already being worked on another project of our team, where we are using SPARQL Construct queries to simulate inference. A Construct query has two parts, a select clause, and a construct clause. By selecting the axiom conditions and constructing the axiom consequences, we effectively obtain inferred triples from the ontology. Afterward, we inject this inferred triples in the ontology with an Insert query. This process could be added to Ontobud, equipping it with a simple and powerful OWL inference mechanism built with the technology already in place.

References

1 Sean Bechhofer, Frank Van Harmelen, Jim Hendler, Ian Horrocks, Deborah L McGuinness, Peter F Patel-Schneider, Lynn Andrea Stein, et al. OWL web ontology language reference. *W3C recommendation*, 10(02), 2004.

2 The Eclipse BlazeGraph framework. `https://rdf4j.org/about/`. Accessed: 2020-04-19.

3 BlazeGraph License. `https://github.com/blazegraph/database/blob/master/LICENSE.txt`. Accessed: 2020-04-19.

4 BlazeGraph Wiki. `https://github.com/blazegraph/database/wiki`. Accessed: 2020-04-19.

5 Dan Brickley, Ramanathan V Guha, and Brian McBride. Rdf schema 1.1. W3C recommendation. *World Wide Web Consortium, February*, 2014. Accessed: 2020-02-04. URL: https://www.w3.org/TR/rdf-schema/.

6 CodeMirror. https://codemirror.net/. Accessed: 2020-04-19.

7 Docker. https://www.docker.com/. Accessed: 2020-04-19.

8 Eclipse Public License – v 1.0. https://www.eclipse.org/org/documents/epl-v10.php. Accessed: 2020-04-06.

9 Express. https://expressjs.com/. Accessed: 2020-04-19.

10 GraphDB Free. http://graphdb.ontotext.com/documentation/free/. Accessed: 2020-04-07.

11 GraphDB Free – About. http://graphdb.ontotext.com/documentation/free/about-graphdb.html. Accessed: 2020-04-07.

12 GraphDB Free – Free Version. http://graphdb.ontotext.com/documentation/free/free/graphdb-free.html. Accessed: 2020-04-07.

13 Aidan Hogan. Linked data and the semantic web standards. In *Linked Data and the Semantic Web Standards*. Chapman and Hall/CRC Press, 2013.

14 Inference. https://www.w3.org/standards/semanticweb/inference. Accessed: 2020-04-23.

15 JWT. https://jwt.io/. Accessed: 2020-04-30.

16 Thorsten Liebig, Vincent Vialard, Michael Opitz, and Sandra Metzl. GraphScale: Adding Expressive Reasoning to Semantic Data Stores. In *International Semantic Web Conference (Posters & Demos)*, 2015. Accessed: 2020-01-09. URL: http://ceur-ws.org/Vol-1486/paper_117.pdf.

17 Frank Manola, Eric Miller, Brian McBride, et al. RDF primer. *W3C recommendation*, 10(1-107):6, 2014. Accessed: 2020-01-01. URL: http://www.w3.org/TR/2004/REC-rdf-primer-20040210/.

18 MongoDB. https://www.mongodb.com/. Accessed: 2020-04-20.

19 Neo4j: A Reasonable RDF Graph Database & Reasoning Engine [Community Post]. https://neo4j.com/blog/neo4j-rdf-graph-database-reasoning-engine/. Accessed: 2020-04-07.

20 Neo4j Overview. https://neo4j.com/developer/graph-database/#neo4j-overview. Accessed: 2020-04-08.

21 Node.js. https://nodejs.org/en/. Accessed: 2020-04-19.

22 NPM bcrypt package. https://www.npmjs.com/package/bcrypt. Accessed: 2020-04-30.

23 PegJS. https://pegjs.org/. Accessed: 2020-04-30.

24 The Eclipse RDF4J framework. https://rdf4j.org/about/. Accessed: 2020-04-06.

25 RDF4J License. https://rdf4j.org/download/#license. Accessed: 2020-04-06.

26 RDF4J Release notes. https://rdf4j.org/release-notes/. Accessed: 2020-04-07.

27 SPARQL. https://www.w3.org/TR/rdf-sparql-query/. Accessed: 2020-04-23.

28 SPARQL 1.1. https://www.w3.org/TR/sparql11-overview/. Accessed: 2020-04-23.

29 SPARQL Grammar. https://www.w3.org/TR/sparql11-query/#rQueryUnit. Accessed: 2020-04-30.

30 Vue. https://vuejs.org/. Accessed: 2020-04-19.

31 Vuetify. https://vuetifyjs.com/en/. Accessed: 2020-04-19.

32 W3 Standards – Ontology. https://www.w3.org/standards/semanticweb/ontology. Accessed: 2020-07-01.

33 Wikidata. https://www.wikidata.org/wiki/Wikidata:Main_Page. Accessed: 2020-04-19.

34 Wikidata Query. https://query.wikidata.org/. Accessed: 2020-04-19.