Investigating Ontology-Based Data Access with GitHub

Yahlieel Jafta[1,3(✉)], Louise Leenen[1,3(✉)], and Thomas Meyer[2,3]

[1] University of the Western Cape, Cape Town, South Africa
2858132@myuwc.ac.za, lleenen@uwc.ac.za
[2] University of Cape Town, Cape Town, South Africa
tmeyer@cair.org.za
[3] Centre for Artificial Intelligence Research (CAIR), Cape Town, South Africa

Abstract. Data analysis-based decision-making is performed daily by domain experts. As data grows, getting access to relevant data becomes a challenge. In an approach known as Ontology-based data access (OBDA), ontologies are advocated as a suitable formal tool to address complex data access. This technique combines a domain ontology with a data source by using a declarative mapping specification to enable data access using a domain vocabulary. We investigate this approach by studying the theoretical background; conducting a literature review on the implementation of OBDA in production systems; implementing OBDA on a relational dataset using an OBDA tool and; providing results and analysis of query answering. We selected Ontop (https://ontop-vkg.org) to illustrate how this technique enhances the data usage of the GitHub community. Ontop is an open-source OBDA tool applied in the domain of relational databases. The implementation consists of the GHTorrent dataset and an extended SemanGit ontology. We perform a set of queries to highlight a subset of the features of this data access approach. The results look positive and can assist various use cases related to GitHub data with a semantic approach. OBDA does provide benefits in practice, such as querying in domain vocabulary and making use of reasoning over the axioms in the ontology. However, the practical impediments we observe are in the "manual" development of a domain ontology and the creation of a mapping specification which requires deep knowledge of a domain and the data. Also, implementing OBDA within the practical context of an information system requires careful consideration for a suitable user interface to facilitate the query construction from ontology vocabulary. Finally, we conclude with a summary of the paper and direction for future research.

Keywords: Ontology-based data access · ontology · data access · relational databases · Git · GitHub

1 Introduction

Information retrieval is a critical process in organizations for extracting insights to achieve strategic organizational objectives. Large enterprises today use sev-

C. Pesquita et al. (Eds.): ESWC 2023, LNCS 13870, pp. 644–660, 2023.
https://doi.org/10.1007/978-3-031-33455-9_38

eral information systems each with its database to store input and functional data [14]. In various domains, clients require access to domain-specific services exported by systems [6]. However, gaining access to the required data in a heterogeneous environment is becoming a challenge due to data access generally being performed by technical experts who translate the requirements of domain experts into the necessary analytical output, creating a bottleneck at scale [17].

The two main ways to handle access to heterogeneous data are procedural and declarative [21]. A procedural methodology is a bottom-up approach where the problem is addressed at the data source level. However, this approach is expensive to maintain and requires updates for each change in the underlying data structure. The declarative approach, a top-down approach, defines a shared conceptualization that is valid for the domain of interest underlying the data sources. This conceptualization is constructed from the intentional level of the application domain terms, which is then linked to the actual data. These terms are then specified to access information [6]. The focus of this paper is based on the declarative approach. To realize such a solution an approach known as Ontology-based data access (OBDA) is advocated for, a technique that utilizes formalized domain knowledge (ontologies) as a suitable formal tool for data access [28]. In the literature, this formalism of domain knowledge is advocated for and applied in the space of problems around data integration and developing intelligent search systems [14]. The focus of this paper is on the latter, however, the literature considers this as a unified problem [14].

This paper applies Ontop, an open-source system that links relational databases with domain ontologies. Ontop transforms the relational database into a virtual Resource Description Framework (RDF) graph [4] that can be queried using SPARQL. This approach offers a powerful way to leverage existing databases for semantic web applications. Our implementation utilizes a GitHub relational dataset from the GHTorrent [11] project. Furthermore, there exists an RDF-linked dataset for GHTorrent called the Semantic Git (SemanGit) [19]. The SemanGit dataset was systematically built by transforming the GHTorrent dataset into linked data, and the SemanGit ontology was subsequently developed.

In this paper, the Ontop OBDA tool is used to illustrate how OBDA enhances the data usage of the GitHub community; Ontop is applied to the GHTorrent dataset and the SemanGit ontology. Section 2 provides the background on OBDA and Sect. 3 discusses related work. Section 4 discusses the Ontop system, the GHTorrent dataset, and the SemanGit ontology (including extensions to the ontology). Section 5 shows the implementation of Ontop on the selected dataset and ontology. Section 6 illustrates query answering with Ontop followed by a discussion in Sect. 7. Finally, we conclude in Sect. 8 with future work emanating from this research.

2 Ontology-Based Data Access

In the last decade, the database research community created the foundation for the utilization of columnar storage [1], allowing for efficient storage and pro-

cessing of large data sets. As a result, relational database management systems (RDMS) have seen considerable growth, especially the wide adoption of database systems offered as cloud services [1]. Ontologies [13] express a shared conceptualization of the domain of interest at a high level of abstraction independent from the data sources. While ontologies are a good candidate for realizing this conceptualization, RDMS are natural candidates for the management of the data layer given the maturity of RDMS.

The Virtual Knowledge Graph (VKG) approach, also referred to in the literature as OBDA, is a well-known view for accessing and integrating data sources [34]. In this approach, the data sources are virtualized through mapping and an ontology, which is presented as a unified knowledge graph that can be queried by end-users using domain vocabulary [34]. When data is queried, the user query is translated over the ontology into SQL queries over the database. The mapping specification layer is responsible for binding the ontology and the data sources. This is achieved by linking the classes and properties in the ontology to SQL views over the data in the database. The ontology in combination with the mappings produces a VKG, which can be queried using SPARQL, the standard query language in the Semantic Web [4]. OBDA systems utilize the mapping specification and Description Logic [2] reasoning to automatically transform queries expressed in terms of the ontology into SQL queries that can be executed on the database.

The OBDA framework consists of an extensional instance, the data source, an intensional schema which is the ontology [32], and the link between the two consisting of a mapping specification.

Definition 1. *Formally, the extensional instance is represented as the* data source ***D*** *conforming to the data source schema* ***S****. The intensional schema is defined as the OBDA specification* ***P*** *= (****O, M, S****) [32] where,*

- ***O*** *is an ontology*
- ***M*** *a mapping from* ***S*** *to* ***O***
- ***S*** *the data source schema*

An OBDA specification **P** is instantiated by a database **D** compliant with the schema **S**. The pair (**P, D**) is referred to as an OBDA instance, or an instance of a VKG. The RDF graph, denoted **M(D)**, is the set of triples produced by combining **M** and **D**. Thus, the exposed virtual RDF graph, denoted $\mathbf{G_{P,D}}$, provides the semantics of an OBDA instance (**P, D**) and comprises the triples derived from the triples in **M(D)** by applying the axioms in **O** [33].

The most fundamental reasoning task in the OBDA approach is query answering over the KG [32]. Query answering is performed by utilizing SPARQL as a query language. A SPARQL query **q** over the OBDA instance (**P, D**) essentially returns the answer to **q** over the KG $\mathbf{G_{P,D}}$, inline with the standard SPARQL semantics [33]. The primary method for query answering in this approach is query reformulation, which prevents physical materialization of the KG $\mathbf{G_{P,D}}$. The SPARQL query **q** expressed over the KG is reformulated into a SQL

query **Q** that can be directly executed on **D** [33]. During the query reformulation process, the SPARQL query **q** is processed through a set of transformations, which include rewriting the query **q** for the ontology **O** and unfolding it inline with the mapping **M**. The answers returned by the SQL query **Q**, after execution on **D**, are returned and transformed into RDF terms based on the mapping **M**. The mapping **M** connecting the ontology **O** to the database is responsible for specifying how the ontology assertions are populated by the data from the source **D**.

3 Related Work

Applications of knowledge graphs are gradually gaining momentum due to their agility and flexibility to apply to various data models [8]. This flexibility enables their application to the integration of heterogeneous sources and schema of data. There has been a lot of attention on converting legacy data to RDF knowledge graphs. Given the wide impact and implementations of relational databases, naturally, the focus shifted in this direction. The two main approaches for this integration were to materialize all data within a given data source as RDF triples or on-the-fly data access using a query language such as SPARQL and to delegate the actual retrieval of the data to the data source engine [26]. Once the data is exposed as VKGs, data can be processed using familiar vocabularies in the form of specific domain ontologies with automated reasoning capabilities.

Using Ontop, Massari et al. [26] apply OBDA to enable data integration of non-relational (NoSQL) data sources with the motivation to fill the gap between NoSQL and the Semantic web. They used Couchbase, a NoSQL document-oriented database with the following components defining the implementation; an OWL Ontology, an Access Interface, mappings, a NoSQL database, a SPARQL to NoSQL query adjustment, and a JSON export [26].

Siemens Energy used OBDA to address data access challenges on large-scale data [17]. The main motivation was the bottleneck for diagnostics in data gathering, which takes up to 80% of the overall time. Finding the right data for analytics is very hard due to the constraints of predefined queries, the complexity of data, the intricacy of query construction, and the limitation of explicitly stated information. Three ontologies, defining the turbine, sensor, and diagnostic data models, based on the Siemens database schemata were developed.

Geologists at the data-intensive petroleum company Statoil frequently use data stored in multiple data sources [16]. Due to the complexities of the data schemata, geologists often require IT personnel for data access support. For example, one of the data stores contains about 3000 tables with about 37000 columns. Given this large data model, query formulation by Statoil geologists is not feasible without the help of IT specialists [16]. OBDA was thus applied to address the data access challenge. Their approach includes an ontology that is connected to the data sources via mappings and query translation between user queries and underlying data sources. The Optique–platform [18] was used.

4 OBDA Tool, Dataset, and Ontology

4.1 Ontop System

In both academia and industry, more than a dozen VKG query answering systems have been developed [33]. To select a suitable query-answering system for our implementation we looked at systems that are open-source with the ability to perform ontological reasoning. Xiao et al. [33] reported on the most important query-answering systems that are compliant with industrial standards and in terms of query performance. The report includes systems that are both open-source and proprietary, irrespective of ontological reasoning capacity. The systems include D2RQ[1], Mastro [5], Morph [29], Ontop [4], Oracle Spatial and Graph[2], Stardog[3] and Ultrawrap [30]. From this list of query answering systems, D2RQ, Morph, and Ontop are open-source, however, both D2RQ and Morph projects do not support ontology inference and have not actively been maintained since January 2015 and June 2022 respectively. Given this, we opted for the Ontop system as the tool of choice.

Ontop has undergone four major releases since its inception in 2009, establishing it as the most mature and state-of-the-art OBDA open-source system [34]. Ontop support the World Wide Web Consortium (W3C) recommendations for SPARQL and the W3C RDB2RDF Mapping Language (R2RML) mappings. During query formulation, Ontop uses "a relational-algebra-type representation" [34] for queries in the intermediate query (IQ) language. Ontop has independent support for each RDMS vendor to produce the desired SQL results. The IQ is converted into the relevant SQL based on the applicable RDMS vendor SQL dialect. For further detail on Ontop, we direct the interested reader to the work of Xiao et al. [34].

4.2 The GHTorrent Dataset

The acquisition and curation of data from software repositories is a typical requirement to support empirical studies on software engineering [11], and GitHub is an attractive source for this as it provides access to its internal public data via a REST API [12]. However, access to the REST API is capped at a request limitation of 15,000 requests per hour per authentication token. Given this limitation, it is quite a cumbersome procedure to extract large amounts of data to support research depending on this data. As of February 2023, GitHub has over one hundred million developers across more than four million organizations contributing to more than three-hundred and thirty million repositories[4], making it a substantial source for software repository data. To address the need for making this data available, the GHTorrent project was established. GHTorrent is an offline mirror of GitHub's event streams and persistent data that is made available to the research community as a service.

[1] http://d2rq.org/.
[2] https://www.oracle.com/database/technologies/spatialandgraph.html.
[3] https://www.stardog.com/.
[4] https://github.com/about.

4.3 SemanGit Ontology

Utilizing an ontology that applies to the domain of the underlying data source is an essential step in OBDA. The GHTorrent dataset falls within the domain of the git version control system (VCS). A VCS keeps track of changes made to a file or set of files over time. Selected files can be restored to their previous state using this feature, promoting easy recovery of files and errors [9]. As part of the investigative procedure to identify the dataset, we had to keep in mind the ontology that will be used. For this, we had the option of developing or re-using an existing ontology for the domain of interest. While investigating a suitable dataset for the research, we found a novel RDF dataset based on the GHTorrent called SemanGit. Based on a git ontology, SemanGit is the first collection of linked data extracted from GitHub [19]. The SemanGit ontology[5] has been identified as a suitable ontology for this research as it was developed and used as the underlying ontology for the RDF-linked dataset created from the GHTorrent dataset [19]. In OBDA, the ontology is used directly over the data source via a mapping specification and keeps the data in its original state. In our work, the SemanGit ontology describes the concepts and relationships in the git domain, while the mapping specification provides a formal mapping between the ontology and the GHTorrent MySQL relational database schema. This enables queries to be expressed in terms of concepts from the ontology using SPARQL. At query time, a SPARQL query is first parsed and analyzed to identify the relevant concepts and relationships from the ontology. Next, the mapping specification is used to translate these ontology concepts and relationships into the corresponding SQL query (tables, columns, and joins) in the underlying relational database schema. Additionally, OBDA can integrate several data sources and should thus not be viewed as a specific data source.

The ontology differentiates between git conventions and provider-specific features. For example, based on git, the author of a commit is represented by a "Name [email]" pair whereas GitHub represents a "commit author" as a user containing additional attributes such as location, country code, creation date [19]. To accommodate this, the ontology was built in a hierarchical structure with the git protocol features forming the base classes, and provider-specific extensions being classes (denoted with the ***"github_"*** prefix) that inherit from the base classes.

We consider the ontology to be a base ontology with ***Primitive*** classes and no ***Defined*** classes. Classes with at least one set of necessary and sufficient requirements are known as defined classes; they have a definition, and every individual who meets the definition belongs to the class. Primitive classes are those that lack any sets of necessary and sufficient requirements [15].

In the SemanGit ontology, ***github_pull_request*** is a subclass of ***pull_request*** which says that if something is a ***github_pull_request*** it is necessarily a ***pull_request***. According to GitHub's REST API, every pull request is an ***issue***, but not all issues are considered pull requests. Given the ontology

[5] https://github.com/SemanGit/SemanGit.

description, if we consider an instance of a ***pull_request***, the knowledge captured is not sufficient to determine that the pull request instance is a member of the class ***github_pull_request*** and that it is an ***issue***. We must alter the conditions to make this possible, by extending the necessary conditions to necessary AND sufficient conditions. This means that the requirements for being a member of the class ***github_pull_request*** are not only necessary but also sufficient to establish that any given instance that satisfies the conditions must be a member of the class ***github_pull_request***. Thus, the classes in the SemanGit ontology are considered to be primitive. The ontology also lacks inverse relations and object property characteristics which make it possible to enrich the meaning of properties [15]. We now outline the extensions made to the ontology. The approach used for extending the SemanGit ontology is based on the methodology defined by Noy et al. [27]. It is an iterative development process that repeats continuously to enhance the ontology. For our purpose, we renamed the classes and properties by removing the underscores and using UpperCamelCase for class names and lowerCamelCase for property names. We also focused on enriching existing class and property definitions. Since we are reusing an existing ontology, the domain *(git protocols)* and scope *(GitHub)* of the ontology is known with the key concepts being defined. Considering this, we are only focusing on the sub-processes related to the extension of class and property definitions. The instances are defined in the underlying database instance. The class extensions applied were minimal. We have converted the class descriptions of ***GithubProject*** and ***GithubPullRequest*** to *definitions*.

- If something is an instance of a ***GithubProject*** then it is necessary that it is a ***Repository*** and it is also necessary that it has exactly 1 owner that is a member of the class ***User***.
  - $GithubProject \sqsubseteq Repository \sqcap \exists hasowner.User \sqcap (= 1 githubHasOwner.User)$
- If something is an instance of a ***GithubPullRequest*** then it is necessary that it is a ***PullRequest*** and it is also necessary that it has **exactly** 1 issue that is a member of the class ***GithubIssue***.
  - $GithubPullRequest \sqsubseteq (PullRequest \sqcap GithubIssue) \sqcap \exists githubPullRequestIssue.GithubIssue \sqcap (= 1 githubPullRequestIssue.GithubIssue)$
- Furthermore, the ***User*** and ***Repository*** classes are disjoint from each other.
  - $User \sqsubseteq \neg Repository$

The following inverse properties were added.

- ***githubOwnerOf*** inverse of ***githubHasOwner***
- ***hasAuthoredComment*** inverse of ***commentAuthor***
- ***hasAuthoredCommit*** inverse of ***commitAuthor***
- ***hasCommittedCommit*** inverse of ***commitedBy***
- ***repositoryHasCommit*** inverse of ***belongsToRepository***

In addition to this, each property was analyzed and updated with a characteristic where applicable. We show a visualization of the ontology in Fig. 1.

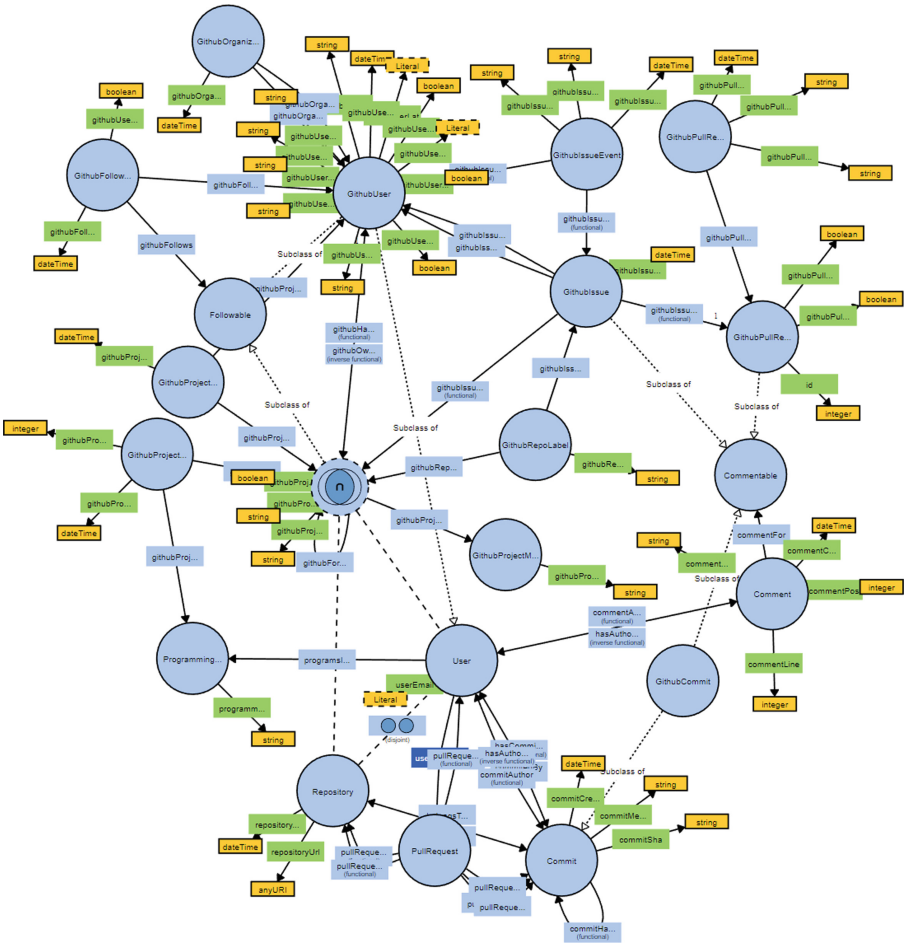

Fig. 1. Ontology visualisation using WebVOWL [24]

5 Mapping GHTorrent to the SemanGit Ontology

To create and manage the mapping assertions for the ontology and database we use the open-source Protégé ontology editor[6] along with the Ontop plugin to enable the management of mappings and querying from within the Protégé editor. We used version 5.5.0 of Protégé and version 4.1.1 of the Ontop plugin. A mapping assertion consists of three components; a unique mapping identifier, a target, and a source. The target is a set of RDF triple patterns defined in the Terse RDF Triple Language (Turtle)[7] syntax that captures the data returned

[6] https://protege.stanford.edu/.
[7] https://www.w3.org/TeamSubmission/turtle/.

by the source, which is a regular SQL query. Figures 2 and 3 shows an example of mapping assertions for the user and commit entity respectively.

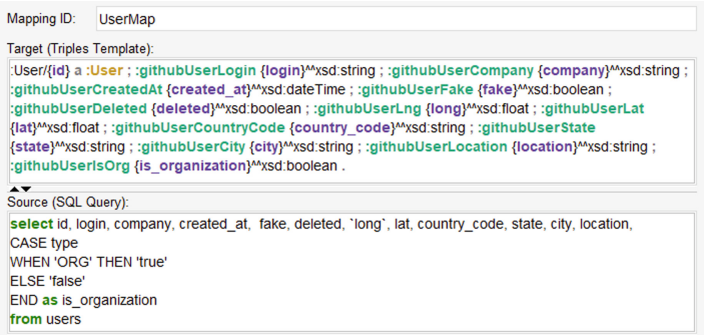

Fig. 2. Mapping assertions for the User entity

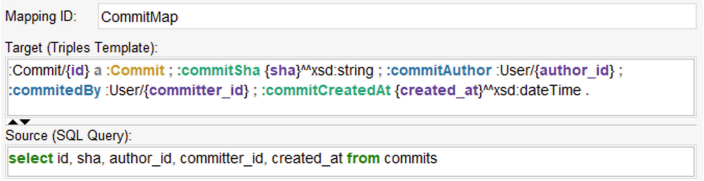

Fig. 3. Mapping assertions for the Commit entity

These assertions construct a part of the knowledge graph (KG) as defined in the target part, by populating the RDF triple pattern answer variables with the corresponding answer in the result set of the source SQL query. The answer variables are enclosed in braces "{" and "}". The "UserMap" mapping assertion populates the ontology User class with the relevant properties to the underlying database instance data. We note that the "UserMap" mapping assertion defines what an organization is considered to be, where an organization according to the dataset is a user database entry with the "type" column populated with the value "ORG". This mapping assertion allows the KG to assert whether a user is an organization based on the boolean value of the *githubUserIsOrg* property that is populated by the computed MySQL column "is_organization". The "CommitMap" mapping assertion populates the ontology Commit class.

6 Querying GHTorrent with SPARQL

To investigate the value of OBDA we performed query answering over the VKG using a select set of queries based on a user not being informed of specific data

encoding schemes and schema structure of the data source. The query experiments were run on a computer with an AMD Ryzen 9 5900X 12-Core Processor running at 3.70 MHz using 32 GB of RAM, running Windows 10 Pro version 21H2. The MySQL database instance was installed on a Gigabyte GP-AG42TB AORUS 2TB M.2 2280 PCI-E 4.0 Solid State Drive with MySQL server version 8.0.

```
SELECT *
WHERE {
  ?organization a :User.
  ?organization :githubUserCountryCode true.
  ?organization :githubUserIsOrg "za".
}
```

Listing 1.1. Select GitHub organizations with country code "za"

```
SELECT v1.‘id‘ AS ‘id1m33‘
FROM ‘users‘ v1
WHERE ('ORG' = v1.‘type‘ AND 'za' = v1.‘country_code‘)
```

Listing 1.2. Generated SQL for listing 1.1

The query in Listing 1.1 selects the GitHub organizations from South Africa (country code "za"). GitHub identifies organizations and users as a **User** entity with a **type** column to distinguish whether an entity is an organization or a standard user. To model this in the ontology, a data property named *"githubUserIsOrg"* is defined with the domain *"githubUser"* and range the "boolean" datatype. In Fig. 2 we show how this property is mapped to the database. Listing 1.2 shows the SQL query translated from the SPARQL query. Here we observe the inclusion of the generated *'ORG' = v1.‘type‘* **WHERE** clause which is a result of the "UserMap" mapping specification in Fig. 2. With this query, we attempt to illustrate that a user does not need to know how the database encodes an "Organization". If the data encoding scheme changes in the database, it will require an update of the mapping assertion which will not affect the SPARQL query if there is no change in the ontology. Here we observe, selecting the organization subset by using the *"githubUserIsOrg"* property in the SPARQL clause (where *githubUserIsOrg* is *true*), unfolds in the *'ORG' = v1.‘type‘* SQL clause after query translation.

```
SELECT ?repo_name ?year (COUNT(?commit) AS ?commits)
WHERE
{
  ?commit :belongsToRepository ?project .
  ?project :githubProjectName ?repo_name .
  ?commit :commitCreatedAt ?date .
  FILTER (?project IN (repo:3905191, repo:12159636))
}
GROUP BY ?repo_name (year(?date) AS ?year)
```

Listing 1.3. Number of commits per year for Angular and React repositories

```
SELECT v7.‘name1m32‘ AS ‘name1m32‘, v7.‘v2‘ AS ‘v2‘,
COUNT(*) AS ‘v4‘
FROM (SELECT v5.‘name1m32‘ AS ‘name1m32‘,
EXTRACT(YEAR FROM v5.‘created_at1m43‘) AS ‘v2‘
FROM (SELECT DISTINCT v1.‘commit_id‘ AS ‘commit_id1m4‘,
v3.‘created_at‘ AS ‘created_at1m43‘, v2.‘name‘ AS ‘name1m32‘,
v1.‘project_id‘ AS ‘project_id1m4‘
FROM ‘project_commits‘ v1, ‘projects‘ v2, ‘commits‘ v3
WHERE (
    (v1.‘project_id‘ = 3905191 OR v1.‘project_id‘ = 12159636)
    AND v1.‘project_id‘ = v2.‘id‘
    AND v1.‘commit_id‘ = v3.‘id‘
    )
) v5
) v7
GROUP BY v7.‘v2‘, v7.‘name1m32‘
```

Listing 1.4. Generated SQL for listing 1.3

In Listing 1.3 we retrieve the number of commits per year for the repositories Angular and React. Angular and React are two popular GitHub repositories. Angular, developed at Google, is a web application development framework that uses Typescript/JavaScript and other languages to create mobile and desktop web applications. React, a JavaScript library for building user interfaces was developed at Meta (formerly known as Facebook). With this example, we illustrate query translation which includes an aggregate function with the commit table and a subset of the columns. We group the results by repository and year to see how the number of commits changed over time. The translated MySQL query can be seen in Listing 1.4.

```
SELECT ?author (COUNT(DISTINCT ?commit) as ?commits)
(COUNT(DISTINCT ?pr) AS ?prs)
WHERE {
  BIND (repo:12159636 AS ?repo)
  ?repo :repositoryHasCommit ?commit .
  ?author :hasAuthoredCommit ?commit .
  ?pr :pullRequestBaseProject ?repo .
  ?pr :pullRequestUser ?author .
}
GROUP BY ?author
```

Listing 1.5. Angular repository contributor commits and pull requests

```
SELECT v6.‘author_id1m7‘ AS ‘author_id1m7‘,
COUNT(DISTINCT(v6.‘id1m28‘)) AS ‘v3‘,
COUNT(DISTINCT(v6.‘commit_id1m5‘)) AS ‘v4‘
FROM (SELECT DISTINCT v2.‘author_id‘ AS ‘author_id1m7‘,
v1.‘commit_id‘ AS ‘commit_id1m5‘, v3.‘id‘ AS ‘id1m28‘
FROM ‘project_commits‘ v1, ‘commits‘ v2,
‘pull_requests‘ v3, ‘pull_request_history‘ v4
WHERE (
```

```
    v1.`commit_id` = v2.`id`
    AND v3.`id` = v4.`pull_request_id`
    AND v2.`author_id` = v4.`actor_id`
    AND 12159636 = v1.`project_id`
    AND 12159636 = v3.`base_repo_id`
    )
) v6
GROUP BY v6.`author_id1m7`
```

Listing 1.6. Generated SQL for listing 1.5

The query above, Listing 1.5 retrieves all the contributors with their total number of commits and pull requests. A **Pull Request** (PR) is a request to merge code changes made on a separate branch of the central repository into the base branch. The database table "*pull_request_history*" stores all the actions associated with a PR, including the user and type of action. We expect to receive results for this query by reasoning over the axioms in the ontology that declare inverse properties, even if we did not include any explicit mapping assertions for the object properties *repositoryHasCommit* and *hasAuthoredCommit*. We use the axioms that *repositoryHasCommit* is the inverse property of *belongsToRepository* and *hasAuthoredCommit* is the inverse property of *commitAuthor* in this scenario.

```
SELECT DISTINCT ?member
WHERE {
  VALUES ?project { repo:27601818 }
  ?member :githubUserFake false .
  ?pr :pullRequestBaseProject ?project .
  ?pr :githubPullRequestMerged true .
  ?pr :pullRequestUser ?member .
}
```

Listing 1.7. Core team members of Vue js project based on Pull Request contributions

```
SELECT DISTINCT v1.`id` AS `id1m51` FROM `users` v1,
`pull_requests` v2, `pull_request_history` v3,
`pull_request_history` v4
WHERE (
    (v1.`fake` = 0) AND v2.`id` = v3.`pull_request_id`
    AND v2.`id` = v4.`pull_request_id`
    AND v1.`id` = v4.`actor_id`
    AND 27601818 = v2.`base_repo_id`
    AND 'merged' = v3.`action`
)
```

Listing 1.8. Generated SQL for listing 1.7

Listing 1.7 retrieves authentic users contributing to the popular GitHub repository Vue[8] based on merges of a **Pull Request** (PR). Authentic users can

[8] https://github.com/vuejs/vue.

own repositories and perform actions such as managing issues, pull requests, and commits. Unauthentic users only show up as commit authors or committers. The *fake* column is used to identify these types of users in the user table. We observe in the translated SQL query, Listing 1.8, the lookup into the "*pull_request_history*" table without explicitly defining it in the SPARQL query (Listing 1.7). This is a result of the mapping specification for the object property *githubPullRequestMerged*, which is populated based on the "merged" action related to a pull request that is stored in the "*pull_request_history*" table. We do note that the generated SQL query contains two self-joins on the "*pull_request_history*" table. The Ontop system uses unique constraints (primary key) for removing self-joins. In the mapping, we are referencing a non-unique constraint column (pull_request_id) for the pull_request_history table. As a test, we observed that when using the primary key in the mapping the self-join was removed.

We repeated each query ten times and took the mean average of the execution time. We compared the execution from within the Protégé SPARQL query editor against running the generated SQL directly in the MySQL Command-line client. We did not notice a major difference in the execution times. Each query was executed against the entire database by selecting "all results" in the Protégé SPARQL query editor. We show the SPARQL query execution time in Table 1.

Table 1. Query execution times

Query	Time (s)
Listing 1.1	9.515
Listing 1.3	0.6877
Listing 1.5	680.8
Listing 1.7	0.2202

7 Discussion

We performed a set of queries to highlight a subset of the features of the OBDA approach. During the execution of the experiments, the feature of querying in domain vocabulary without the need to understand the underlying database data encoding and schema as well as utilizing the ontology axioms during query executions does stand out. The results look positive and can assist various use cases related to GitHub data with a semantic approach. The ontology enables a more precise understanding of the relationships between different data elements, allowing for intelligent data querying. However, the practical impediments we observe are in the "manual" development of a domain ontology and the creation of a mapping specification which affects scalability.

The query volume, the size and complexity of the ontology, and the stability and performance of the underlying system all have an impact on scalability.

And while it is possible to scale OBDA systems in a production environment, in contrast to traditional database systems, it is a complex endeavor that requires deep knowledge to develop and maintain domain ontologies and mapping specifications that do not suffer semantic loss between the original data and associated ontologies. The mapping specification connecting the ontology to the database involves writing individual queries that must be consistent with the vocabulary of the ontology for each database table and column [4]. And while the development and maintenance of ontologies is a well-established topic with considerable research [31], the engineering of mapping specifications is still an emerging technology. Given this complexity, mapping engineering is a demanding procedure. Several mapping engineering methodologies and tools have been proposed to address this challenge. Xiao et al. [33] group the contributions into two categories: mapping bootstrappers and editors. A mapping bootstrapper attempts to automate a mapping specification for a relational data source. However, the generated ontology and mappings are data source specific, whereas a domain ontology aims at being used across multiple data sources within a domain. Mapping editors like the ontology editor Protégé provide an environment for mapping engineering but do not support features such as syntax highlighting and require deep-level knowledge about the underlying mapping language [33]. We also mention Ontopic Studio[9], a more recent no–code mapping editor to link databases and data lakes with knowledge graphs. For further detail, we direct the reader to https://ontopic.ai/en/ontopic-studio/.

Traditional relational database systems on the other hand have lower complexity, are scalable, and have defined best practices to achieve good performance in production given the level of maturity. In comparison to existing large systems and our experimental observation, OBDA currently falls short in terms of complexity, cost-effectiveness, and maturity. However, OBDA allows for a more detailed understanding of the connections that exist between data pieces. Thus the trade-off between scalability and the reasoning capacity of OBDA needs to be taken into consideration. Also, we note that actualizing OBDA within the context of an information system requires careful consideration for the implementation of a suitable user interface to facilitate the SPARQL query construction from ontology vocabulary, where users of such a system are querying from a client-facing user interface and not writing SPARQL queries. In this work, we assumed that a user performing queries is familiar with GitHub terminology and that the ontology is modeled as close as possible to this domain to enable query construction from the vocabulary terms. Another approach would be to look at the field of Natural Language Processing to assist in query formulation, such as integrating machine learning algorithms and knowledge representation to query knowledge graphs in natural language. This is however out of the scope of this research, we refer the interested reader to [10,22].

Given these impediments, the research in this field is very active. Such as semi-automating ontology development using an approach called Ontology Learning (OL), where machine learning techniques are applied to represent

[9] https://ontopic.ai/en/.

knowledge from heterogeneous data sources. Recent work in this area includes various proposals to apply OL in the scope of relational databases [3,20,23,25]. In the work by Calvanese et al. [7], the authors proposed an algorithm to automatically detect and map a relational schema to ontology mapping patterns.

The artifacts of this research can be found at this link, which includes the extended ontology and the full set of mapping specifications for each database table. We highlight that the GHTorrent dataset is not currently up to date, and the last data dump was released in March 2021. We also note that the website is not available anymore, however, the content is available on the GitHub repository. We are not aware of any future updates to the GHTorrent dataset.

8 Conclusion and Future Work

This paper describes the application of OBDA, specifically the use of the Ontop tool, on a well-defined open-source dataset called GHTorrent which is based on the GitHub platform. OBDA involves connecting an ontology to a data source using a mapping specification. The SemanGit ontology was used as the underlying ontology for this work, which we have extended. We documented the mapping procedure and demonstrated query answering using SPARQL. OBDA's querying in domain vocabulary, combined with reasoning over the ontology's axioms, shows promising results. There are opportunities for this work to be extended and applied for specific use cases applicable to GitHub and OBDA. This includes publishing the extended ontology and making this work publicly available to the GitHub community via an interface and API endpoint for further evaluation. Maintenance of the extended ontology will be ongoing and can take several directions depending on the scope of use cases. Furthermore, this research can facilitate the broader domain of artificial intelligence in the area of knowledge extraction from heterogeneous data. We thank the reviewers for their comments and suggestions. The authors used the Protégé editor, supported by grant GM10331601 from the National Institute of General Medical Sciences of the US National Institutes of Health.

References

1. Abadi, D., et al.: The Seattle report on database research. ACM SIGMOD Rec. **48**(4), 44–53 (2020). https://doi.org/10.1145/3385658.3385668
2. Baader, F., Horrocks, I., Lutz, C., Sattler, U.: Introduction to Description Logic. Cambridge University Press, Cambridge (2017). https://doi.org/10.1017/9781139025355
3. Ben Mahria, B., Chaker, I., Zahi, A.: A novel approach for learning ontology from relational database: from the construction to the evaluation. J. Big Data **8**(1), 1–22 (2021). https://doi.org/10.1186/s40537-021-00412-2
4. Calvanese, D., et al.: Ontop: answering SPARQL queries over relational databases. Semant. Web **8**(3), 471–487 (2017). https://doi.org/10.3233/SW-160217

5. Calvanese, D., et al.: The MASTRO system for ontology-based data access. Semant. Web **2**(1), 43–53 (2011). https://doi.org/10.3233/SW-2011-0029
6. Calvanese, D., De Giacomo, G., Lembo, D., Lenzerini, M., Poggi, A., Rosati, R.: Ontology-based database access. In: SEBD, pp. 324–331 (2007)
7. Calvanese, D., et al.: ADAMAP: automatic alignment of relational data sources using mapping patterns. In: La Rosa, M., Sadiq, S., Teniente, E. (eds.) CAiSE 2021. LNCS, vol. 12751, pp. 193–209. Springer, Cham (2021). https://doi.org/10.1007/978-3-030-79382-1_12
8. Calvanese, D., Lanti, D., De Farias, T.M., Mosca, A., Xiao, G.: Accessing scientific data through knowledge graphs with Ontop. Patterns **2**(10), 100346 (2021). https://doi.org/10.1016/j.patter.2021.100346
9. Chacon, S., Straub, B.: Pro Git. Springer, Heidelberg (2014)
10. Chen, Y.H., Lu, E.J.L., Ou, T.A.: Intelligent SPARQL query generation for natural language processing systems. IEEE Access **9**, 158638–158650 (2021). https://doi.org/10.1109/ACCESS.2021.3130667
11. Gousios, G.: The GHTorrent dataset and tool suite. In: Proceedings of the 10th Working Conference on Mining Software Repositories, MSR 2013, Piscataway, NJ, USA, pp. 233–236. IEEE Press (2013). https://doi.org/10.5555/2487085.2487132
12. Gousios, G., Spinellis, D.: GHTorrent: GitHub's data from a firehose. In: 2012 9th IEEE Working Conference on Mining Software Repositories (MSR), pp. 12–21. IEEE (2012). https://doi.org/10.1109/MSR.2012.6224294
13. Gruber, T.R.: A translation approach to portable ontology specifications. Knowl. Acquis. **5**(2), 199–220 (1993)
14. Gusenkov, A., Bukharaev, N., Birialtsev, E.: On ontology based data integration: problems and solutions. In: Journal of Physics: Conference Series, vol. 1203, p. 012059. IOP Publishing (2019). https://doi.org/10.1088/1742-6596/1203/1/012059
15. Horridge, M., Jupp, S., Moulton, G., Rector, A., Stevens, R., Wroe, C.: A practical guide to building "OWL" ontologies using protégé 4 and co-ode tools edition1. 2. The University of Manchester, vol. 107 (2009)
16. Kharlamov, E., et al.: Ontology based data access in Statoil. J. Web Semant. **44**, 3–36 (2017). https://doi.org/10.1016/j.websem.2017.05.005
17. Kharlamov, E., et al.: Semantic access to streaming and static data at Siemens. J. Web Semant. **44**, 54–74 (2017). https://doi.org/10.1016/j.websem.2017.02.001
18. Kharlamov, E., et al.: Optique: ontology-based data access platform (2015)
19. Kubitza, D.O., Böckmann, M., Graux, D.: SemanGit: a linked dataset from `git`. In: Ghidini, C., et al. (eds.) ISWC 2019. LNCS, vol. 11779, pp. 215–228. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-30796-7_14
20. Lakzaei, B., Shamsfard, M.: Ontology learning from relational databases. Inf. Sci. **577**, 280–297 (2021). https://doi.org/10.1016/j.ins.2021.06.074
21. Lenzerini, M., Daraio, C.: Challenges, approaches and solutions in data integration for research and innovation. In: Glänzel, W., Moed, H.F., Schmoch, U., Thelwall, M. (eds.) Springer Handbook of Science and Technology Indicators. SH, pp. 397–420. Springer, Cham (2019). https://doi.org/10.1007/978-3-030-02511-3_15
22. Liang, S., Stockinger, K., de Farias, T.M., Anisimova, M., Gil, M.: Querying knowledge graphs in natural language. J. Big Data **8**(1), 1–23 (2021). https://doi.org/10.1186/s40537-020-00383-w
23. Liao, C., Wu, Y., King, G.: Research on learning OWL ontology from relational database. In: Journal of Physics: Conference Series, vol. 1176, p. 022031. IOP Publishing (2019). https://doi.org/10.1088/1742-6596/1176/2/022031

24. Lohmann, S., Link, V., Marbach, E., Negru, S.: WebVOWL: web-based visualization of ontologies. In: Lambrix, P., et al. (eds.) EKAW 2014. LNCS (LNAI), vol. 8982, pp. 154–158. Springer, Cham (2015). https://doi.org/10.1007/978-3-319-17966-7_21
25. Ma, C., Molnár, B.: Use of ontology learning in information system integration: a literature survey. In: Sitek, P., Pietranik, M., Krótkiewicz, M., Srinilta, C. (eds.) ACIIDS 2020. CCIS, vol. 1178, pp. 342–353. Springer, Singapore (2020). https://doi.org/10.1007/978-981-15-3380-8_30
26. El Massari, H., Mhammedi, S., Gherabi, N., Nasri, M.: Virtual OBDA mechanism Ontop for answering SPARQL queries over Couchbase. In: Saidi, R., El Bhiri, B., Maleh, Y., Mosallam, A., Essaaidi, M. (eds.) ICATH 2021. LNDECT, vol. 110, pp. 193–205. Springer, Cham (2022). https://doi.org/10.1007/978-3-030-94188-8_19
27. Noy, N.F., McGuinness, D.L., et al.: Ontology development 101: a guide to creating your first ontology (2001)
28. Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M., Rosati, R.: Linking data to ontologies. In: Spaccapietra, S. (ed.) Journal on Data Semantics X. LNCS, vol. 4900, pp. 133–173. Springer, Heidelberg (2008). https://doi.org/10.1007/978-3-540-77688-8_5
29. Priyatna, F., Corcho, O., Sequeda, J.: Formalisation and experiences of R2RML-based SPARQL to SQL query translation using Morph. In: Proceedings of the 23rd International Conference on World Wide Web, pp. 479–490 (2014). https://doi.org/10.1145/2566486.2567981
30. Sequeda, J.F., Miranker, D.P.: Ultrawrap: SPARQL execution on relational data. J. Web Semant. **22**, 19–39 (2013). https://doi.org/10.1016/j.websem.2013.08.002
31. Staab, S., Studer, R. (eds.): Handbook on Ontologies. IHIS, Springer, Heidelberg (2009). https://doi.org/10.1007/978-3-540-92673-3
32. Xiao, G., et al.: Ontology-based data access: a survey. In: International Joint Conferences on Artificial Intelligence (2018). https://doi.org/10.24963/ijcai.2018/777
33. Xiao, G., Ding, L., Cogrel, B., Calvanese, D.: Virtual knowledge graphs: an overview of systems and use cases. Data Intell. **1**(3), 201–223 (2019). https://doi.org/10.1162/dint_a_00011
34. Xiao, G., et al.: The virtual knowledge graph system Ontop. In: Pan, J.Z., et al. (eds.) ISWC 2020. LNCS, vol. 12507, pp. 259–277. Springer, Cham (2020). https://doi.org/10.1007/978-3-030-62466-8_17