World Academy of Science, Engineering and Technology
International Journal of Computer and Information Engineering
Vol:13, No:5, 2019

Parallel Querying of Distributed Ontologies with Shared Vocabulary

Sharjeel Aslam, Vassil Vassilev, Karim Ouazzane

Open Science Index, Computer and Information Engineering Vol:13, No:5, 2019 publications.waset.org/10010391/pdf

Abstract**—**Ontologies and various semantic repositories became a convenient approach for implementing model-driven architectures of distributed systems on the Web. SPARQL is the standard query language for querying such. However, although SPARQL is well-established standard for querying semantic repositories in RDF and OWL format and there are commonly used APIs which supports it, like Jena for Java, its parallel option is not incorporated in them. This article presents a complete framework consisting of an object algebra for parallel RDF and an index-based implementation of the parallel query engine capable of dealing with the distributed RDF ontologies which share common vocabulary. It has been implemented in Java, and for validation of the algorithms has been applied to the problem of organizing virtual exhibitions on the Web.

Keywords**—**Distributed ontologies, parallel querying, semantic indexing, shared vocabulary, SPARQL.

I. Introduction

THE conventional approach for developing Web applications uses a combination of ad-hoc techniques which utilize various methods, technologies and tools tailored specifically for the World-Wide Web. This approach is prone of many drawbacks due to the lack of unification, the struggle with the complexity and the crippling limitations, and it is not an accident that the recent DevOps movement heavily relies on the use of more agile methodologies for development to cope with the problem. Model-driven approaches, on the other hand, rely on the direct use of models on different levels of abstraction. The ontological engineering as universal modeling technique, equally suitable for the Web as well as for other media, is a promising alternative.

The ontologies serve two different purposes in software development. On one hand, they help conceptualizing the solutions by providing rich modeling capabilities, strictly based on formal logics and thus, they help avoiding logical errors in the design. On the other hand, the ontologies provide a firm basis for implementing intelligent solutions which incorporate elements of AI. Because of this, we can conclude that the use of ontologies in the design and implementation of model-driven architectures of software system is of key importance. RDF as the *lingua franca* of the Semantic Web is of special importance for achieving quality, universality and productivity. For the purpose of building semantically rich distributed information systems using model-driven architecture, it is necessary to have a suitable query language for querying the ontological repositories. Although there are several candidates – SPARQL, SERQL, RDQL and RQL, SPARQL is suggested by the World Wide Web Consortium (W3C) as a standard language for querying RDF repositories. It has all the elements which are essential for querying RDF data sets, including distributed ontologies [10]. Unfortunately, most of the existing APIs which support SPARQL, such as Java Jena library, for example, do not provide support for querying distributed ontologies. This is obviously due to the complications related to the need for concurrency management. Additional complexity comes from the need to maintain multiple vocabularies. This is a serious restriction on the possibility to use ontological approach, since the distribution is often an essential requirement in developing enterprise applications.

Our research is focused on a restricted version of distributed ontologies with shared vocabulary. Unlike the general case of distribution in such ontologies, there is no need to maintain multiple vocabularies and, as a result, the concurrency is reduced to an enumerated number of patterns for which the concurrency management can be achieved easily on a case-by-case basis. The core idea is to reduce the concurrency issue to the concurrency of a simple RDF triple as the only semantic representation within the ontology. Thanks to this, the concurrency management can be guaranteed by utilizing the semantic equivalence of the dependencies of the three components of the RDF tuples in the local ontologies, which are always in the format <Subject, Predicate, Object>. This article introduces a complete framework for utilizing distributed ontologies in the process of developing intelligent distributed applications on the Web. It also illustrates the application of the framework for organizing online virtual exhibitions using information extracted from multiple participating museums. Although the validation of the framework is tailored to the Web, its use is not limited to that media and can be easily adopted in a more general case of model-driven architectures (MDA).

II. Current State of Research

Distributed ontologies occur in multiple cases where the modelled resources are physically distributed across multiple locations. The typical solution which database systems use for dealing with the physical distribution is based on some sort of

Sharjeel Aslam was a PhD student at the School of Computing and Digital Media of London Metropolitan University. He is currently with the University of Northumbria, UK (e-mail: sharjeel.aslam@northumbria.ac.uk).

Vassil Vassilev is a Reader in AI and Cyber Security. He is currently Head of the Cyber Security Research Centre of London Metropolitan University, UK (corresponding author; phone +447762794887, e-mail v.vassilev@londonmet.ac.uk).

Karim Ouazzane is a Professor in Knowledge Transfer. He is currently with the School of Computing and Digital Media of London Metropolitan University, UK (e-mail: k.ouazzane@londonmet.ac.uk).

replication. However, such a purely syntactic mechanism cannot be used in distributed ontologies due to their rich semantics, which require more sophisticated mechanisms for synchronization of the operation and the data across multiple locations. In this section, we will review some applications of the ontological approach, specifically focusing on the effect of the distribution.

Regardless the particular area of application, the first problem of distributed ontologies which needs to be resolved is the parallel search [1]. *Ding's* for example introduced a semantic flash search engine, which is reported to deliver the top 50 retrieved results from the Google search [2]. But most intelligent semantic search engines do not perform very well when precision and low recall are needed. The recognition of the intension behind the search plays also an important part and some end-user search engines have incorporated user sentiment analysis as part of the solution to improve the precision and reduce the recall rate [3]. However, this solution is not applicable to the parallel search of distributed ontologies, since it may eliminate important local findings and thus, to reduce the precision.

Another problem related to the distributed search is the necessity to maintain explicit representation of the different meanings of the search terms in different ontologies and to support contextual mapping of these meanings, known as *semantic disambiguation*. This might be critical in intelligent applications since the limited terminological knowledge may lead to formulation of wrong queries [5]. Typically resolving the semantic ambiguity of the search terms requires a thesaurus such as **WordNet**, but only a few search engines present as an option terminological search with multiple meanings [4]. Some researchers prefer to use numerical ranking of the context of use, but although more computationally efficient, this method may cause serious errors when the Semantic Web applications require definite meaning based on pure logic and merely linguistic information [6].

The distribute systems with ontological models also experience integration problems. They occur within the layer responsible for merging the local ontologies into an integrated global ontology [7]. In principle, two alternative methods can be used to merge the ontologies:

1. Combining the ontologies into a solitary global ontology.
2. Keeping the local ontologies isolated from each other and maintaining of communication protocol.

In both cases, the ontologies must be brought into a sharing contract which explicitly specifies the shared resources.

The evolving environment coupled with the absence of information about possible upgrades of the ontologies makes the results of the queries less trustworthy due to the inter-dependencies between the predicates [8]. As a consequence, this may lead to partial or full disintegration of the systems.

The evolution of the ontologies can affect server performance as well due to the different workload it takes to find the response to queries. The query optimization needs to produce more robust execution plans because cardinality gauges change too rapidly [9]. At the same time, early scheduling may turn out to be extremely perplexing.

The mapping of the terms used within different ontologies is an important mechanism for maintaining the global consistency and integrity so it needs special attention. In most of the cases this is addressed by the use of the so-called "semantic bridge". The RDFT system [10] delineates a little tool to portray the various mappings amongst the RDF repositories. The key concept in this approach is the bridging, which is described using a separate "semantic bridge ontology". Beyond RDF, this concept is further exploited in the other languages of the semantic languages "cake" – OWL [11] and KIF [12].

Some of the above problems can be avoided if limiting to distributed ontology with shared vocabulary. In such a case, the terminology used within different ontologies is automatically synchronized with the other ontologies and any changes or extensions of one ontology do not lead to problems in the other.

III. Proposed Framework for Querying Distributed Ontologies with Shared Vocabulary

SPARQL is to RDF ontologies what SQL is to the relational databases but on a higher level of abstraction. The result of the execution of the SPARQL query is an RDF graph consisting of all triplets which match the stated condition in the query. They can be used for producing reports directly out of the repository to perform semantic disambiguation using the semantic dependencies between different terms or as a knowledge representation for intelligent applications.

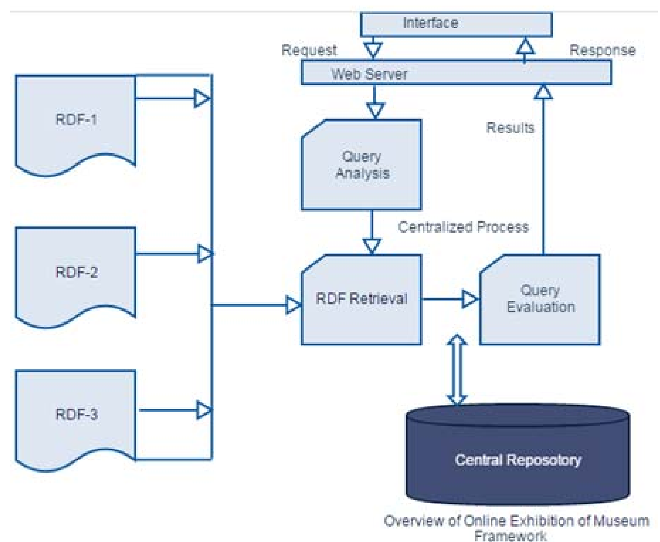

Fig. 1 Framework for Concurrent Query Processing in Distributed RDF Ontologies with Shared Vocabulary

Conceptual architecture of our framework is shown on Fig. 1. The ontology is distributed amongst a number of physically dispersed repositories **RDF-1**, **RDF-2** … **RDF-N**, while the **Central Repository** contains the shared vocabulary and the supporting information which allows to reformulate the original SPARQL queries as independent subqueries which

Open Science Index, Computer and Information Engineering Vol:13, No:5, 2019 publications.waset.org/10010391/pdf

can be executed against the local repositories. The three principle components of the framework are the **Query Analyzer**, responsible for finding semantic equivalent subquery in each of the repositories, the **RDF Triple Retriever,** which performs SPARQL query processing against each separate repository, and the **Query Evaluator**, which performs aggregation of the results of the subquery execution into semantically equivalent global response to the original SPARQL query.

In the subsequent sections of this article, we will present systematically the methods and the algorithms used by the **Query Analyzer** and the **Query Evaluator** during runtime execution of the global SPARQL queries, as well as the supporting mechanisms of the **Central Repository** working offline as an advanced preparation of the shared vocabulary for common use in real time.

IV. Virtual Exhibitions Scenario

In order to validate the proposed framework, we have developed a dedicated scenario for organizing of virtual exhibitions using information from several museums. Each of the participating museums in the above scenario represents the information about its own exhibition funds in a local RDF repository built using a common vocabulary of terms. The virtual exhibition is organized around a set of queries in SPARQL format which are inherently concurrent and require parallel execution and synchronization. An RDF schema of the vocabulary is shown on Fig. 2.

The above vocabulary contains three main categories of information:

- Museum exhibits with classification of the items;
- Craftsmanship with classification of the art forms;
- Place of origin with geographic location; and,
- People and their role in relation to the exhibits (artists, curators, critics, etc.).

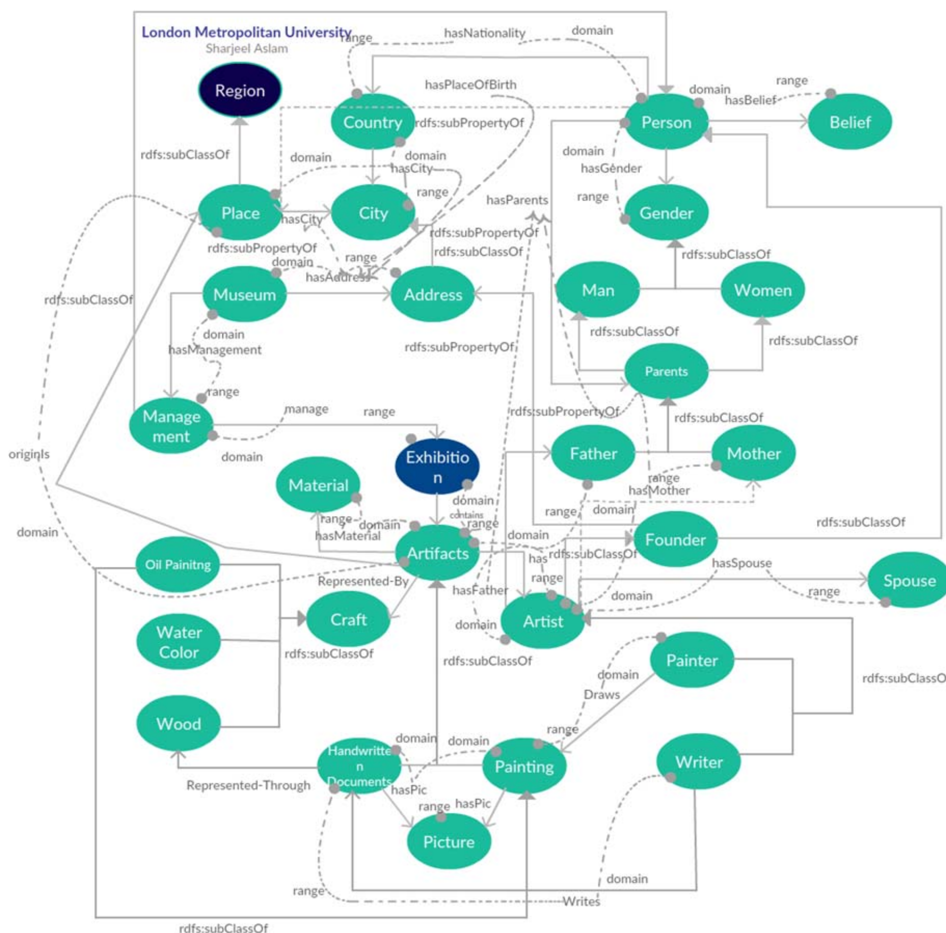

Fig. 2 RDF Schema of the shared vocabulary

In total, the shared vocabulary contains 30 classes. Albeit being small, it is entirely sufficient for the purpose of testing. There are numerous cases of potential concurrency which appear in the distributed ontology:

- Identical classes with different descriptions in different repositories;
- Identical classes with different relations in different repositories;
- Identical classes with different inheritance in different repositories;

Open Science Index, Computer and Information Engineering Vol:13, No:5, 2019 publications.waset.org/10010391/pdf

- Superclass in one repository, sub-classes in other repositories;
- Different classification of individual objects in different repositories, etc.; and,
- Different relations of individual objects in different repositories, etc.

An excerpt of the RDF repository which contains the above model is as follows:

```
<Place is-a Region>
<Place has City>
<Country has City)
<Museum hasAddress Address>
<Address is-a City>
<Museum hasManagement Management>
<Management manage Exhibition>
<Exhibition contains Artifacts>
<Artifacts hasMaterial Material>
<Artifacts represented-By Craft>
<OilPainting is-a Craft>
<Watercolour is-a Craft>
<Wood is-a Craft>
<Painting is-a Artifacts>
<HandWrittenDocuments is-a Artifacts>
<HandWrittenDocuments represented-through Wood>
<Painting hasPic Picture>
<HandWrittenDocuments hasPic Picture>
<Artifacts has Artist>
<Artist is-a Founder>
<Founder is-a Person>
<Painter is-a Artist>
<Writer is-a Artist>
<Artist hasSpouse Spouse>
<Artist hasFather Father>
<Artist hasMother Mother>
<Father is-a Parents>
<Mother is-a Parents>
<Parents is-a Man>
<Parents is-a Women>
<Man is-a Gender>
<Women is-a Gender>
<Person hasGender Gender>
<Person hasBelief Belief>
<Person hasNationality Nationality>
<Person hasPlaceOfBirth Place>
```

V. RDF Algebra

The RDF algebra is a formal semantic model for interpretation of the syntactic operations in RDF exactly as the relational algebra is a formal model for interpreting the SQL operations. As such, it includes the well-known relational algebra operations (*selection, projection, join*) plus the additional operations which correspond to non-relational syntactic expressions (*aggregation, generalization* and *specialization*). It will be used for interpretation of the SPARQL queries and is a main vehicle for implementing the parallel query engine.

A. Projection

$$\boldsymbol{\pi}_{\left[\substack{S?\\O?}\right]}(\boldsymbol{source})$$

$\boldsymbol{\pi}$ denotes the operation which takes two parameters $\boldsymbol{S}$ and $\boldsymbol{O}$ and applies to the set denoted as $\boldsymbol{source}$. In RDF ontologies, the source (the schema) consists of triplets with three elements: *Subject, Predicate* and *Object.* Projection $\boldsymbol{\pi}$ therefore will extract information about the subjects and the objects of the triplets in the schema.

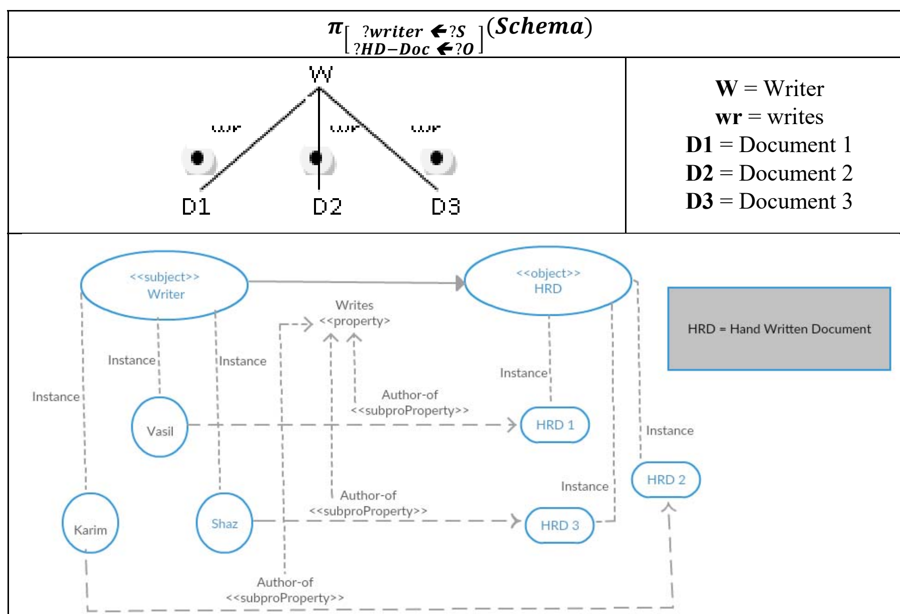

Fig. 3 A list of resources about writer and handwritten documents

B. Selection

$$\boldsymbol{\sigma}_{[condition]}(\boldsymbol{source})$$

$\boldsymbol{\sigma}$ denotes an operation which filters the triplets in the source schema by selecting only the ones which meet the parametric

Open Science Index, Computer and Information Engineering Vol:13, No:5, 2019 publications.waset.org/10010391/pdf

condition. The condition itself can be represented as a combination of arithmetic, comparison and Boolean operations meaningful for the schema.

C. Join

$$\pi_{[?X\,,?Y]}\boldsymbol{\sigma}_{[\boldsymbol{condition}]}(\boldsymbol{source})$$
$$\bowtie$$
$$\pi_{[?X\,,?Z]}\boldsymbol{\sigma}_{[\boldsymbol{condition}]}(\boldsymbol{source})$$

$\bowtie$ denotes the join. It combines triplets from a single or multiple source schemes according to the requested query. The variables *?X* represent subject and *?Y* represent object. Selection operator $\boldsymbol{\sigma}$**,** will be used to filter the joined schemes using the specified conditions, while the projection operation π allows to constructs the joins using the selection of subjects, objects and predicates from the joined schemes.

D. Generalization

Generalization is the process of extracting common characteristics from one or more classes and combining them into characteristics of a more general class, their superclass. Generalization operator, $\boldsymbol{Gen}$ will be used to find the parent super-class of a given **?class** within the source schema. But since both the *subject* and *the object* in RDF are classes which can have super-classes or subclasses, the operation must be transitive. Therefore, it can be used to get the entire hierarchy of classes and subclasses up to a specific level.

E. Specialization

Specialization is the reverse operation of generalization, i.e., it finds the subclasses of an existing class within the same schema.

$$\boldsymbol{Spec}_{(\boldsymbol{rdfs}:\boldsymbol{class}(?\boldsymbol{class}),\boldsymbol{n-level})}(\boldsymbol{Schema})$$

VI. Distributed SPARQL Query Processing

The query processing in our framework combines offline preparation by indexing of the shared vocabulary with real-time manipulations of the original query in four steps – translation of the original query, splitting of the original query into sub-queries, local execution and global aggregation.

A. Vocabulary Indexing

The vocabulary indexing plays an important role in the implementation of the parallel query processing engine for distributed ontologies with shared vocabulary. It facilitates the process of translation of the global SPARQL queries into local sub-queries by providing semantic correspondences between the elements of the triplets. At the same time, it allows to implement the search algorithms in a more efficient manner.

The indexing procedure is executed against the shared vocabulary offline. It is based on the assumption that the vocabulary is identical in all ontologies and is performed incrementally as the ontologies are loaded. The index is represented internally as a table with the following index keys:

- Data sources
- Subject roles
- Predications
- Object roles
- Generalizations
- Specializations
- Filtering conditions

An example of full indexing is shown in Table I. These indices play different roles. The first key, for example, is used by the algorithm for translating the original query into subqueries, while the next three keys are used to speed up the search by navigation, and the last two keys are used for substitutions of semantically equivalent classes.

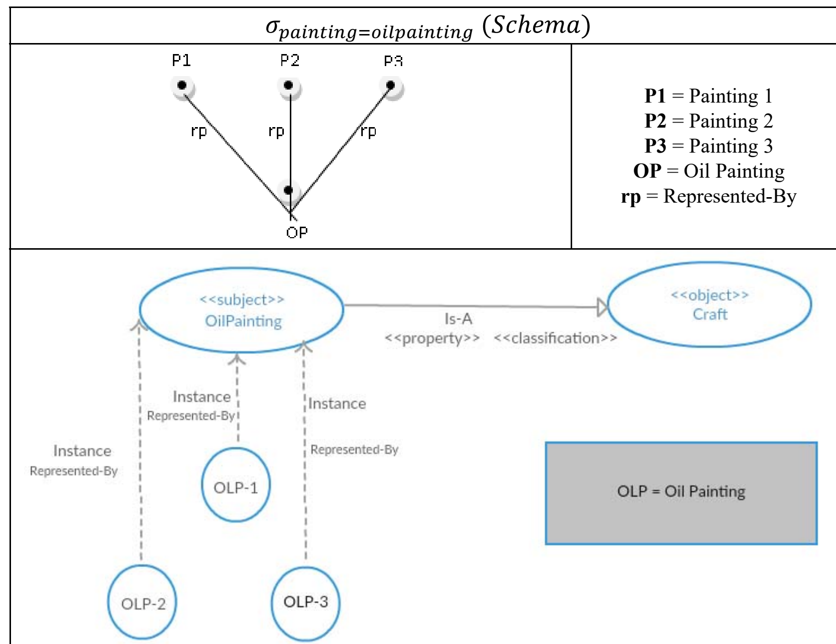

Fig. 4 All paintings of Oil Painting

Open Science Index, Computer and Information Engineering Vol:13, No:5, 2019 publications.waset.org/10010391/pdf

Open Science Index, Computer and Information Engineering Vol:13, No:5, 2019 publications.waset.org/10010391/pdf

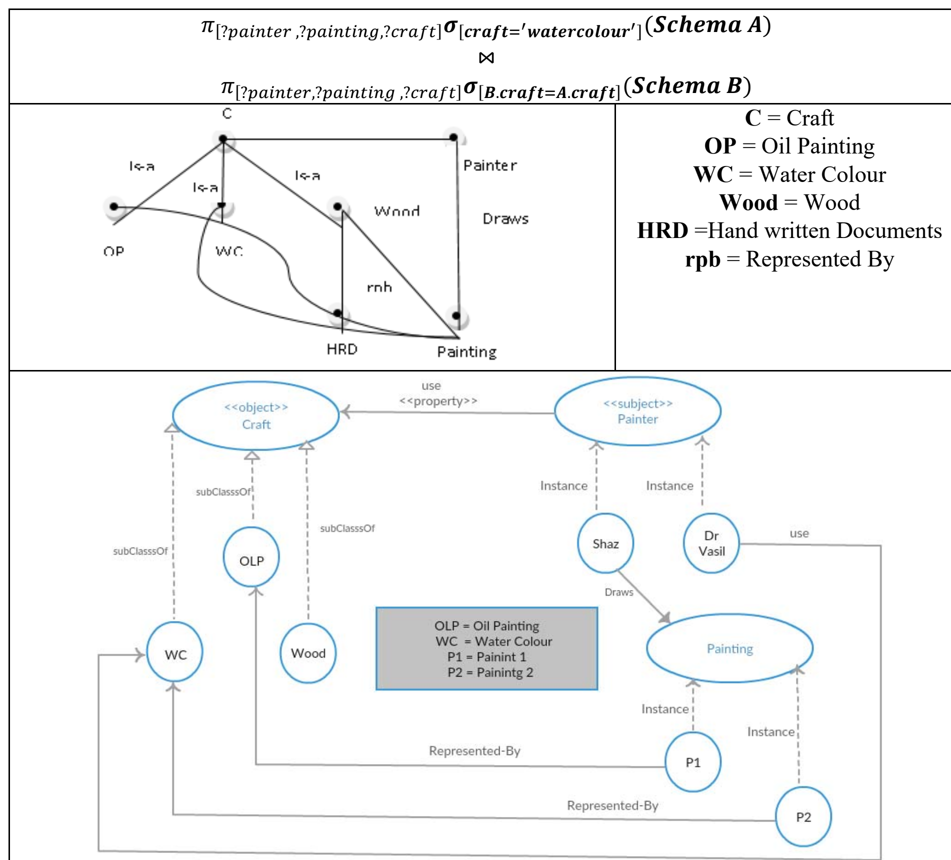

Fig. 5 All paintings of painters where the craft used is watercolor

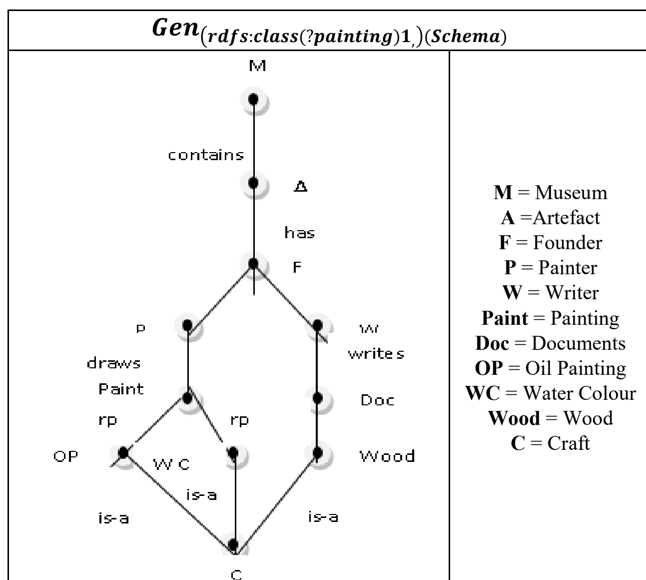

Fig. 6 Hierarchy of painting at level 1

The algorithm for indexing of the vocabulary is a simple loop which registers the relevant information associated with each new RDF triplet. It is executed by a dedicated utility written in Java. It uses extensively, the Jena API for traversing the vocabulary. The index table itself is maintained by the **Central Repository**.

TABLE I
INDEXING OF THE MUSEUM ONTOLOGY

Subj	Pred	Spec	Gen	Cond	DS
?place	hasCity	Craft	Address Place Painter Oilpainting		Ds1 Ds2
?painter	draws				Ds1 Ds2
?country	hasCity				Ds1 Ds2
?museum	hasAddress				Ds1 Ds2
?museum	hasArtefacts				Ds1 Ds2
?artist	hasCountry				Ds1 Ds2

B. Query Translation

The translation step is necessary to prepare the split of the general query. It converts the SPARQL queries into semantic operations. As an example, let us consider the following SPARQL query:

```
PREFIX m:<http://allahm.museum.org/museum#>
SELECT ?museum, ?exhibition, ?management
WHERE {
    m:?museum rdf:hasManagement m:?management
    m:?management rdf:manages m:?exhibition
 }
```

Our algorithm translates this query into the algebraic

expression,

```
(π (?museum, ?exhibition, ?management)
  (⋈
    (σ
       m:?museum rdf:hasManagement m:?management
       m:?management rdf:manages m:?exhibition
    )
  )
)
```

which can be used for splitting the query into independent subqueries.

C. Subquery Extraction

After the translation step, the general SPARQL query which may contain concurrency is converted into a set of semantically equivalent subqueries, which can be executed against the separate ontologies independently. This is done by accounting the semantic indices of the common vocabulary. The general algorithm for splitting the original query is shown in the appendix. For example, the query for finding the address of the museums which translates into the algebraic expression:

```
(π (?museum, ?place, ?city, ?address)
  (⋈
     (σ
        Spec(rdfs:class(?city)1)
                m:?place rdf:hasCity m:?city,
             m:?museum rdf:hasAddress m:?address,
             m:?museum "science"
     )3
  )
)
```

using the indices is split into two separate subqueries which are to be executed against the two data sources independently, as shown in Table II.

TABLE II
SPLITTING THE QUERY INTO SUBQUERIES

Subject	Predicate	Object	Spec	Gen	Cond
?place	rdf:hasCity	?city	City		
?museum	rdf:hasAddress	?address			science

D. Response Aggregation

The general algorithms for executing the local subqueries and for aggregating the global response are shown in the Appendix. The execution is straightforward and is based on the SPARQL interpreter of the Jena API. The generated local results are further aggregated by combining the separate RDF triplets to produce the final response to the global query.

VII. Conclusion

This article presents a complete framework for executing concurrent queries against distributed RDF ontology: an RDF algebra for formal description of the SPARQL queries, a system architecture for software implementation and a set of algorithms for the main software components. The framework has been implemented in Java using the popular API for RDF and SPARQL, Jena. It has been tested successfully using a working scenario for organizing virtual exhibitions from museum repositories where it shows excellent performance.

Our approach is based on the assumption that the local RDF repositories of the distributed ontology share a common vocabulary. While it looks restrictive at first glance, this assumption is quite natural and does not create any practical difficulty. It allows to avoid the problem of clashes due to different naming standards and the need for additional mapping of the names and types between different repositories.

Our current implementation makes extensive use of the semantic indexing. It is a separate step during which the common vocabulary is indexed incrementally as the individual repositories of the distributed ontology are loaded. This step is executed entirely offline during the preparatory stage of organizing the virtual exhibition. Thanks to the hashed data structure used to represent the index cash in the case of adding more repositories, it is necessary to re-index the vocabulary only against the new repositories, while the repositories which have been indexed previously do not need re-indexing.

One restriction of our current implementation is the limitation of the taxonomic relations to degree 1 only, i.e. currently we look only for super-classes and subclasses without accounting the transitivity. In our immediate plans is a possible extension which will account the transitivity of the taxonomic relations. This will require only recursive amendments of the algorithms for search without changing the rest of the framework.

Another potential for further development is the query optimization. It can be based on the extensive development of the optimization techniques developed for the relational databases.

Appendix

Algorithm 1: Translating SPARQL query into algebraic expression

```
Create function transformToAlgebricForm
 which receive queryString and model
 Create Query of given sparql query string
 using create method of QueryFactory.
 Create the pattern element of created Query
 Create Op object to compile the query
 Optimize the Algebra expression
 Initialize variable varMap as HashMap
 Create object of NodeTransform with varMap
 Call transform method to get query into algebraic form
End function
```

Algorithm 2: Converting a SPARQL query into subqueries

```
Create function generateSubQry
 which receive Linked Hash Map of triplePath
 and set of Strings containing required model names
 Declare variable parentModels as a set of Model
 and assign keySet of ModelMap
 Declare variable modelTripleMap
 with key Model and value LinkedHashSet of TriplePath
 Declare variable triplesForModel as
LinkedHashSet<TriplePath>
 Begin for loop
 get the key of entry into tripleName
```

Open Science Index, Computer and Information Engineering Vol:13, No:5, 2019 publications.waset.org/10010391/pdf

```
 get the value of entry into set of String
 if modelSet contains modelname
 add triplename to triplesForModel
 End if
 save the model and triplesForModel to map modelTripleMap
 End for loop
End function
```

Algorithm 3: Executing SPARQL subquery locally

```
Create function runqueryonModel
 which takes modelTripleMap and modelCollection as input
 Declare parentModel
 Declare variable Map<String, String>subQueryDetails
 Begin for loop for each model existingModel in
parentModel
 get model name of existingModel and prefix of
existingModel
 execute the query using queryExecution engine
 to receive the resultset of executed query
 if ResultSet has next element
 add modelname and query to subQryDetails
 split the modelname and store it into array fname
 create file with "subquery" appended to fname
 End if
 End for loop
End function
```

Algorithm 4: Aggregating the local results

```
Create function runQueryonModels(List<Model>
modelCollection,
 String queryFinal)
 get substring of query with index of select and last
index
 Declare Function
ReadableIndex.createReadableIndex(FileFilter)
 Declare variable Map<String, String>subQryDetails and
 initialize Map<String,String> subQryDetails = new
HashMap<>();
 Begin for loop for each model existingModel in
parentModel
 get model name of existingModel
 get the prefix of ExistingModel
 execute the query using queryExecution engine
 get the resultset of executed query
 if ResultSet has next element
 add modelname and query to subQryDetails
 split the modelname and store it into array fname
 create file with "subquery" appended to fname
 create fileoutputstream of above mentioned file
 write result to file using ResultSetFormatter
 End if
 close fileoutputstream and queryEngine
 End loop.
 get Map<String,String> subQryDetails – list of
subqueries
 combine subqueries with string append operation
 get the list of models
 iterate over models and execute query using queryEngine
 create fileWriter and write query results to csv file
End function
```

REFERENCES

[1] G. Gardarin, H. Kou, K. Zetourni et al. SEWISE: An Ontology-based Web Information Search Engine (http://subs.emis.de/LNI/Proceedings/Proceedings29/GI-Proceedings.29-9.pdf).

[2] D. Ding, J. Yang, Q. Li, L. Wang, and W. Liu, "Towards a flash search engine based on expressive semantics," in Proceedings of WWW Alt.'04 New York, 2004, pp. 472-473.

[3] C. Lee, Alan Liu, "Toward Intention Aware Semantic Web Service Systems," scc, vol. 1, pp.69-76, 2005 IEEE International Conference on Services Computing (SCC'05) Vol-1, 2005.

[4] M. Tumer, A. Shah, and Y. Bitirim, 'An Empirical Evaluation on Semantic Search Performance of Keyword-Based and Semantic Search Engines: Google, Yahoo, Msn and Hakia', in *Fourth International Conference on Internet Monitoring and Protection, 2009. ICIMP '09*, 2009, pp. 51 –55.

[5] Z. Gefu and H. Zhao-hui, 'Design of a Semantic Search Engine System for Apparel', in *2010 International Conference on E-Business and E-Government (ICEE)*, 2010, pp. 1414 –1417.

[6] Z. ZhiHao, H. JiPing, D. Ting, and W. Yu, 'Semantic Web Service Similarity Ranking Proposal Based on Semantic Space Vector Model', in *2012 Second International Conference on Intelligent System Design and Engineering Application (ISDEA)*, 2012, pp. 917 –920.

[7] H. S. Pinto, A. G´omez-P´erez, and J. P. Martins. Some issues on ontology integration. In *Proceedings of the Workshop on Ontologies and Problem Solving Methods (IJCAI-99)*, 1999.

[8] Bruijn, J. d. (2010). RIF RDF and OWL Compatibility. Retrieved from W3C Recommendation: http://www.w3.org/TR/2010/REC-rif-rdf-owl-20100622/.

[9] Y. Kalfoglou and M. Schorlemmer. Ontology mapping: the state of the art. *The Knowledge Engineering Review*, 18(1):1–31, 2006.

[10] B. Omelayenko. RDFT: A Mapping Meta-Ontology for Business Integration. In *Proceedings of the Workshop on Knowledge Transformation for the Semantic for the Semantic Web at the 15th European Conference on Artificial Intelligence (KTSW2002)*, pages 77–84, Lyon, France, 23 July 2002.

[11] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini, and H. Stuckenschmidt. C-OWL: Contextualizing ontologies. In D. Fensel, K. Sycara, and J. Mylopoulos, editors, *Lecture Notes in Computer Science*, volume 2870, pages 164–179. Springer, June 2003.

[12] D. Calvanese, G. De Giacomo, and M. Lenzerini. A framework for ontology integration. In *Proc. of the First Semantic Web Working Symposium*, pages 303–316, 2007.

Open Science Index, Computer and Information Engineering Vol:13, No:5, 2019 publications.waset.org/10010391/pdf