Indian Journal of Science and Technology, *Vol 9(19), DOI: 10.17485/ijst/2016/v9i19/90024, May 2016*
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

A Read and Write Enhanced Platform - R2R Ingression for RDF-to-RDB

V. Sitha Ramulu[1]* and B. Raveendra Babu[2]

[1]Department of Computer Science and Engineering, SBIT, Khammam, India;
vsitaramu.1234@gmail.com
[2]Department of CSE, Dean-Administration and Finance, VNRVJIET, Hyderabad, India;
raveendrababu.b@vnrvjiet.in

Abstract

Background and Objectives: Relational Databases have got such a great importance in the current global enterprises, such that they are widely and extensively used for storing and managing data. The objective includes the comparison techniques with older approaches and present approach depicts best accessibility to read/write access to the data. The data semantics are not directly imposed on the relational schema. In fact, they are direct towards the application level. **Methods/Statistical Analysis:** There are several mapping techniques involved by many semantic web technologies and several ontologies that enhance data utilization among many applications. The techniques also embed data reusability for many number of times across community boundaries, large enterprises etc. The current strategies depend on the Read only access either through Linked data or SPARQL. The implemented approach underwent the test cases with various instances using interfaces. The interfaces falling under semantic web frameworks for accessing of data by these approaches include RDF2GO: ChangeSet, Sesame and Jena. **Findings:** The feasibility factors for conversion of relational data schema to the RDF do not give satisfactory remarks. Hence, to increase the feasibility scope for conversions, we used and adopted an approach called "Mediation Approach" for converting RDF's to RDB's. In the current context we present an extensible platform called RDB-to-RDF Ingression (R2RI) that supports Read/write access to the data residing in the relational schema. The entitled platform uses the technique of "encapsulation" as the translation logic during translation process in the core layer. This translation logic proves itself to be the foundation for the rest of the interfaces existing in the interface layer. At later stages, we exposed the general concepts about the architectural patterns, mappings of RDB-to-RDF and emphasizing special focus on our mediation platform "R2RI" duly enhancing a protocol to provide parsing feedback protocol to the client. We also present the prototype implementation process followed by its performance calculations.

Keywords: Relational Databases, Resource Description Framework (RDF), Semantic Web

1. Introduction

Relational Databases claimed great importance in the current global enterprises, such that they are widely and extensively used for storing and managing data. The data semantics are not directly imposed on the relational schema. In fact, they are direct towards the application level. The RDF and ontologies initially build a parsing layer which is called as semantic layer. This semantic layer brings all the processed data to its semantic stage. In the current database systems, it is not always recommended to convert all applications data in to RDF, since, few applications and their data depend on relational representation. Replacing or adapting of such applications data would require greater effort in terms of migration technique. Keeping in view of this point, we followed an approach called Mediation Approach that immediately performs the translation process of all semantic web requests on demand. Hence, this mediation approach profoundly knocked the outcome of converting all relational applications. In supplement to

*Author for correspondence

the above, the mediation approach exhibits various merits such as its scalability, security, transaction support and performance. SPARQL[1], which is a querying language in the field of semantic web technologies, is enraged for querying the RDF data. There are also several familiar data access interfaces in the semantic web frameworks such as RDF2Go, Jena, Linked data and Sesame. The Semantic web has a mere shortfall with the DML (Data Manipulation Language). Therefore, SPARQL/Update[2], a querying language was incorporated by the World Wide Web Consortium (W3C) to enhance the support of DML in the upgraded version of SPARQL 1.5. In due course of time, many other several approaches stepped in the scenario for RDF update process. E.g., Guo, ChangeSet. The SPARQL 1.1 shall enforce data access interface which can be represented as *Graph*store HTTP*protocoll*[3]. So, in precise the mediator for translating RDB-to-RDF should not support only single interface but, must be liable to handle multiple interfaces across the databases. The client may be quiet confusing sometimes when a read-only query throws no result and obviously when a write request also promotes an error out of non- processing of requests. At such circumstances these requests are not handled, and in return a parsing protocol providing feedback is provided to the client in RDF in the specific format.

The specific highlights in this paper include the Mediation Approach for translation of RDB-to-RDF using R2RI. The proofs of the mapping languages show bidirectional support for both read/write access. At later stages, we expose the general concepts about the architectural patterns, mappings of RDB-to-RDF and emphasizing special focus on our Mediation Platform R2RI duly enhancing a protocol to provide parsing feedback protocol to the client. This feedback acknowledges the client to change/append the invalid requests as per his desire.

The World Wide Web Consortium (W3C) has identified the significance of mapping languages and especially in RDB-to-RDF. In this connection, it has enforced RDB2RDF Incubator Group (XG) to study the process for recognition. XG recommends[4] to define a base for mapping language i.e., RDB-to-RDF. Pushback is a project from the W3C community that is used to write back the changes increased in the generation of RDF data by the wrapper of API's 2.0. RDB's are supposed to be published on the semantic web, and the approach used was D2R3. It enhances the browsing facility of the relational data as RDF through URI availability. D2R's main initiative is to provide inter linkage of datasets in the web and expressed in the form of RDF. The open link software Virtuoso features views in RDF in the relational data and specifically uses the metadata schema mapping language for converting teams in ontology to the concepts in schema.

This approach finalizes the SPARQL as complimentary language. The views in the RDF are not updatable since the views are of read only tagged queries. R_20[5] is also categorized as one among the Meta Data Declarative Language to differentiate the mapping between the ontologies and database schema. R_20 is exclusively targeted when there are very few similarities existing between the database model and the ontologies. It reached the expectations of having better expressiveness, balancing complicated mappings over another whose model is more specific or well-structured to the other. It enhances write support from RDF based annotated HTML forms (RD Forms). *Pushback* is primarily concentrated on the web applications where modification of RDB's are not supported in the project.

The paper is organized as follows: The section 2 illustrates the descriptive work done in the scope of RDB-to-RDF mapping. The section 3 comes forward with examples in defining the RDB-to-RDF mapping languages with R2RI. The section 4 entails the architectural patterns and prototype implementations about our Mediator Approach. The section 5 clearly talks about the parsing/semantic protocol called feedback protocol to the RDF client. The section 6 highlights the extensible approach of our used platform that works with the state of art refinement factors. The section 7 summates with experimental setup explaining the performance issues of software evolution. The section 8 in the present paper concludes with a conclusion.

2. The RDB-to-RDF Ingression Mapping

The mediation needs a mapping from the concepts hailing in RDB schema to the terms characterized in the ontology. However, many of the investigations claimed that they are not suitable for write access to the data in the RDF. The already existing mapping languages are supposed to be extended further by supplementing vast information to provide support for write access. View Update problem should be identified by the mapping language[6]. Mostly, all approaches today employ SQL views over the schema

to explain the mappings. This reaps high expressiveness, besides raises view update problem (write access is an impractical issue). Keeping in view about the view update problem, we have designed and introduced our mapping language called R2RI which greatly overcomes the drawback of view problem by adding and facilitating write access to the data. R2RI is an extended version of mapping approach explained[7]. R2RI is very rich in associating the use cases as clearly explained[8], and mapping the normalized relational schema.

We have defined the definitions of mappings with various examples in our R2RI approach, providing the bidirectional relations successfully. Hopefully, the definitions are not deceived by the view update problem. Thus, R2RI enhances read/write support for the schema. The subject of R2RI explains with selected examples which are in the form of RDF-based syntax. The namespace prefixes is widely used in the areas of given examples: R2RI is the mapping language which maps based on RDF syntax and is explained in detail with the selected examples. The name spaces are also part of the examples and explained in *Dublin core* and *friend of friend* projects.

The listing 1(a) explains about the *TableMap* that maps the database table to the corresponding class. All the table maps in the given mapping language adopt the mapping functions. The table map represents the concerned information in the table. At line 2, it has the name of the table. At line 3, it states about the ontology class it is mapped to. At line 4, an URI pattern is described to evolve URI instances on the basis of the table attributes and are clearly specified with in double percentage signs (%%id%%, id is the primary key attribute). The lines at 5 to 8 denote the list of Attribute maps in the *TableMap.*

```
1   a) ex:author a r2ri:TableMap;
2       r2ri:hasTableName «author»;
3       r2ri:mapsToClassfoaf:Person;
4       r2ri:uriPattern «http://.../author%%.id%%»;
5       r2ri:hasAttributeex:author_id,
6           ex:author_email,
7           ex:aut.hor_firstname,
8           ex:aut.hor_lastname.
9
10  b) ex:author_email a r2ri:AttributeMap;
11      r2ri:hasAttributeName «email» ;
12      r2ri:mapsToObject.Propertyfoaf:mbox;
13      r2ri:hasConstraint [ a r3m:NotNull ].
14
15  c) ex:publication_author a r2ri:LinkTableMap;
16      r2ri:hasTableName «publication_author»;
17      r2ri:mapsToObjectPropertydc:creator;
18      r2ri:hasSubjectAttribute ex:pa_publi.cation;
19      r2ri:hasObjectAttribute ex:pa_author.
```

Listing 1: Mapping Examples

The listing 1(b) demonstrates an example for the *AttributeMap* stating that an attribute of a database is mapped to its corresponding property in the ontology. All the Attribute maps in the given mapping language adopt the mapping functions. At line 11, in the database schema the *attribute Map* comprises of the name of the attribute. At line 12, it describes the property of the ontology it is being mapped. At line 13, the attribute map holds particular information relating to the constraints defined on that attribute (For example - a not null constraint) in the map. R2RI foreign key, r2ri:Default, r2ri:primary-key and r2ri:NotNull are the constraints supported by the attribute map. The listing 1(c) presents the *LinkTableMap* that depicts the mapping of *LinkTable* to its corresponding ontology property. All of the sets in *LinkTable* get mapped by the mapping language while adopting the mapping functions. At line 16, the *LinkTableMap* describes about the name of the link table in the schema. At line 17, it describes about the ontology property it is mapped. The link table comprises of two foreign key attributes that pin points to the tables with N:M relationships. At lines 18 and 19 the foreign key attributes with N:M relations are called as attribute maps that facilitate the names to every attribute, the mapping relationship from subject to object and enables the references of foreign keys to the respective tables.

3. The R2R Ingression Architectural Patterns and its Implementation Process

The major asset of the R2RI is to facilitate a platform to the RDF that supports read/write to the data. The platform widely supports a number of interfaces and enhances future scope of development for accessing the data. The R2RI platform uses the technique of encapsulation so as to hide the translation logic of RDB-to-RDF in to its core operations. The encapsulation technique used by the platform eradicates the repeated and overflows of translation process during the mapping. This has improved the flexibility of the data access by adding additional data

interfaces. On using CRUD[13] operations, we came across using three varieties of operations on the data access for semantic web. Query operations on one triple pattern, operation pertaining to adding group of triples to the existing data and operation pertaining to deleting group of triples from the existing data. In precise, these core operations do not support any kind of implementation for data access. They honorably ignore the requests like atomicity and performance.

The architectural pattern of R2RI is depicted in the Figure 1. The architecture is split up in to two layers. The upper layer is called as the "interface layer" which exhibits the functional aspects of the R2RI for the data access. The interface layer cannot be accessed directly either by the application domains or by the network services. The lower layer is called as the "core layer" and it shoulders responsibility for the actual translation process of RDB-to-RDF.

4. The Core Layer in the Architecture

The core layer is highly responsible for all the interactions with in the database schema. The layer is also responsible in implementing the basic operations during

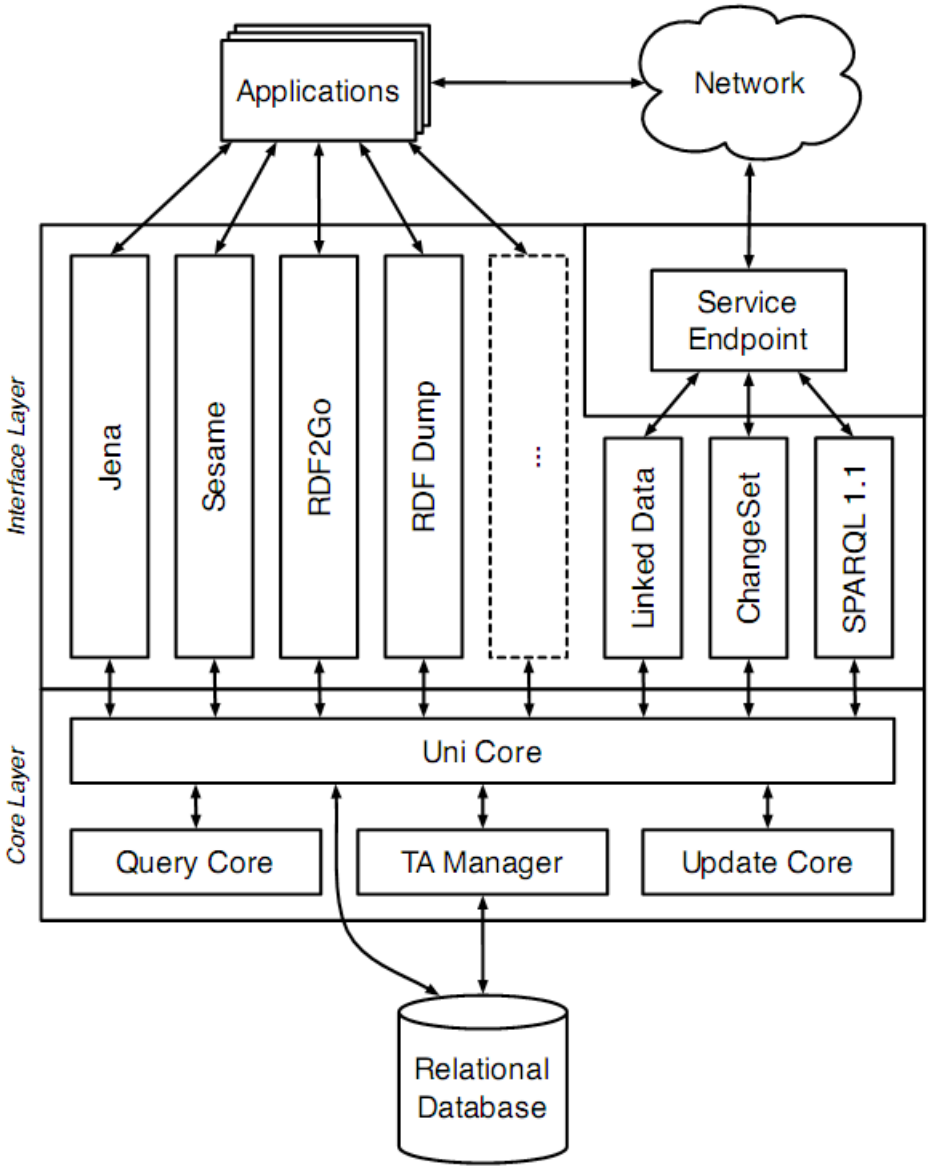

Figure 1. Architectural pattern of R2RI.

the translation process from RDB-to-RDF. The core layer in the architecture comprises of unicore, the query core, the update core and the TA manager. The unicore model in the architecture facilitates the usages of API's existing from the core layer to the interface layer. The unicore works like a controller to the rest of the three modules. It also waves its controlling effect over the encapsulation technique used for data hiding. The API's from the unicore uses two methods of requests. The primary method is used to query a request and the secondary method is used to update request. The query method uses the query core. The data access interfaces collect all the individual requests pertaining to a single transaction and unites these requests and submits all together at once. The unicore integrates the update core and the query core which are responsible in collecting all the individual transaction requests made on the database schema. The database management and its transactions are controlled over by the TA manager. The TA manager looks after the commitment, start and rolling back the transactions after it receives the instructions from the unicore module. The major modules in the core layer comprises of Update core and the Query core that rely emphasis on the translation logic and implementation process of RDB-to-RDF.

a. **Query Core:** The query core module in the core layer performs the root primary operation of querying the triples which are in the form of patterns. Depending upon the type and style of the pattern, the query core starts translating all the triple patterns in to multiple SQL queries or to a single query in the given mapping. For example - a pattern requesting for an object with given predicate and subject generates only on SQL query. Similarly, a pattern consisting of a predicate and subject associating with an object obviously generates several SQL queries in multiple tables of the schema.

The translation process of triples in to the SQL queries are briefly explained from the algorithms 1 to 4. First, it is very important to identify and differentiate the triple patterns those have a strong subject (as shown in algorithm 1).

Algorithm 1. S,P,O Triple pattern translations

```
1. if sub is a variable then
2.     queries := translateTable(t, pre, obj)
3. table  := identifyTable(sub)
4. else
5. for all table in getTables() do
6. queries :=translateTable(t, pre, obj)
7. end for
8. end if
9. return queries
```

At line 3 from the algorithm 1, a strong subject (sub) from the triple pattern is used to identify the table (t) that it perfectly matches (through URI). At lines from 5 to 7, a variable subject is generally employed for generating query for the table it has mapped to.

Algorithm 2. Translation table for (t, pre, obj)

```
10. if pre is a variable then
11. queries :=translateAttribute(t, attrib, obj)
12. attrib:=identifyAttrib(t, pre)
13. else
14. Attrib:=getAttrib(t)
15. queries:=translateAttrib(t, attrib, obj)
16. end if
17. return queries
```

At algorithm 2, we now test whether the predicate pattern is variable or strong. At line 3 in algorithm 2 a strong predicate is translated to its corresponding attribute. But, we cannot predict for a variable predicate to which attribute it is mapping to. Therefore, at line 5 to 6, it is very essential to merge all the attributes those were mapped in the query.

Algorithm 3. Translation Attributes for (t, attrib, obj)

```
1. if obj is a variable then
2.     value := NULL
3. else
4.     value := extractvalue(obj)
5. end if
6. queries:=assembleQuery(t, attrib, value)
7. return queries
```

At algorithm 3, we clearly separate the strong and variable objects. The value of the database is expelled from the strong objects as per the mapping defined. At line 2, if the object found is variable it is immediately marked as null.

Algorithm 4. Congregate Query for (t, attrib, value)

```
1. select :=t.getPrimarykey()
2. from :=t
3. for all attrib in attributes do
4. if value is NULL then
5. select := attribute
6. where:=getCondition(attrib, value)
8. end for
9. return buildQuery(select, from, where)
```

Algorithm 4 at line 6, we finally conclude all the queries by assembling them. The affected table has the primary key which is added to the variables. For instance—the usage of select clause at line1 is a primary key. Hence, the table is attached to 'from clause'. At lines 3 to 7, the iteration process continues around the attributes by adding conditions, such as 'where clause'. At lines 4 and 5, the usage of select list comes in to enforcement by adding the attributes to the values, if the values are found to be null. At line 6, large numbers of multiple conditions are clubbed together with the usage of 'or' operator. At line 8, the final query is generated by using 'from', 'where' and 'select' parts after which they are returned. Once the translation process is finished, all the resulting queries are put in hidden format using the technique of encapsulation in TripleIterator that rides the utility package of java with interfaces 'java.util.iterator' to facilitate a standard iteration process for the results of queries generated in triple patterns. The complete process is done in a sequential manner. Sometimes there may be only simple active query, which is evaluated as its first SQL query and the outcome of the query is widely used in the development of result triples. The result set is generated, and new SQL query is further evaluated.

This resulted in two major merits. Primarily, memory is optimized to a greater extent since it allows only single SQL Resultset object. Secondly, if the user is interested in viewing all results simultaneously, it enhances only few subsets of the triples. These circumstances show greater impact on the performance issues by explicitly presenting fragments of data in the table. The look ahead iterator is the name given to the TripleIterator as it implements the generation of new triples by using next() operator. The process is continued in a sequential manner by caching all the triples. This approach is taken in to consideration because of the variants arousing between the JavaIterator and the SQL ResultSet. The left over results are further verified by HasNext() method. The HasNext() is not featured by the SQL API. Suppose, if there exists only left over results, these are carried out by look-ahead iterators to establish a connection to the API gap and increases their performance by eradicating unnecessary cursor movements from the leftover results.

b. Update Core: All the existing triples in the update core undergo translation process, and finally converted to SQL DML statements. The update core translation processes uses the specialized algorithmic approach[9] for translating delete and insert data operations compared to the SPARQL/Update[11]. The update core operation of insert and delete remains same except with the defragmented SQL statement. Firstly, all the triples possessing similar subject are grouped, so as to treat the same subject record in the schema. This feature enhances the translation process of triples easily because of its similarity. The second step involves in notifying the affected table through Subject URI. The third step involves in mapping the triples so as to check that all triples fall within the integrity constraints of the relational databases. Based on the mapping definitions the fourth step generates various SQL[12] statements for deleting and adding set of triples. The predicate belonging to each and every triple is further translated in to its equivalent attribute. The existing object is represented as data value. Finally, all the similar subject group triples are executed as per the request with enhancing the atomicity.

4.1 Data Access Interface Layer

The interface layer for accessing the data acts as a bridge between the semantic application and the R2RI. It is precisely recognized through set of interfaces applied in the applications by means of service end points which are discussed in the forthcoming concepts. The Jena Linked Data Interface acts as direct and indirect interfaces. The individual interfaces on the R2RI are specifically used to translate all the operations pertaining to the interface and the outcomes are returned back in specific interface format. These interfaces are implemented in the core layer because of its light weight. Few among the light weight interfaces are Sesame, RDF2GO, RDFDump, ChangeSet, Linked Data and Jena[7] are currently used in the R2RI translations.

Here, we consider Jena which is a light weight interface as an example and is explained in precise. The Jena API has two different layers to interact with the RDF. They are *GraphAPI* and the *ModelAPI*. The lower layer is represented by the *GraphAPI*. The main task of *the GraphAPI* is to retrieve the triple and store them. It enhances the facility to delete, add and locate the triples. The alternate layer *ModelAPI* is at the user end and facilitates proper environment to work with the data of RDF. The two API's are described in such a way that they can be extended to add many triples to the database schema. Jena by default has the ability to implement many triples for storing on the disk/memory. For R2RI with Jena Interface, we

stated an implementation process through a graph called GraphAPI. Jena Interface does not require any extension enhancement because it is capable in creating a standard model from existing *GraphAPI*.

The Figure 2 describes the Jena Interface with appropriate UML diagrams. Jena Interface facilitates the core layer to be interacted very frequently in the R2RI. The main API class in the R2RI graph comprises of deleting, finding and adding methods for triples.

These methods are the basic operations which are used to map on the R2RI. The JenaIterator in the R2RI graph develops an Iterator Interface from the final method. The iterator interface acts like a wrapper to our triple Iterator. The transaction handler which is a class in the Jena Interface facilitates support for complete transactions. The triples which are supposed to be deleted or added in existing database schemas are implemented by our 'R2RITransactionHandler'. The added or deleted triples are further transferred to the core of the R2RI for complete transactions. The triple representations in the Jena Interfaces for R2RI are converted by using the Util class that comprises of several formal methods.

4.2 The Service Endpoint

The R2RI facilitates accessing of data interfaces through network by means of server applications called service endpoint. Implementing Handler class gets registered at the end point to access data interfaces through network. The network services for the applications work through HTTP. Whenever a request is made, and the request matches its corresponding target, then the query string hits the respective interfaces. The requested strings those do not match the appropriate target are considered to Linked data requests by default.Java servlet technology is widely used by the service endpoint.

5. Parsing Feedback Protocol

The gap existing between the RDF and the relational models shows great impact during translation process of write requests from RDF to data base schema. The query comprises the instances or ontology terms which are supposed to be mapped to the schema but cannot be perfectly mapped. In such circumstances the query is processed and can return zero results. Any way if the write requests with any non-mapping ontologies are existing then it results an error. These under specified write request with zero results are considered with respect to constraints. It is not always possible to ignore such write requests for the mapping. In either of the approaches explained above, the client may not be aware of the issue why the request is not processed. This happens more predominantly if the client is purely

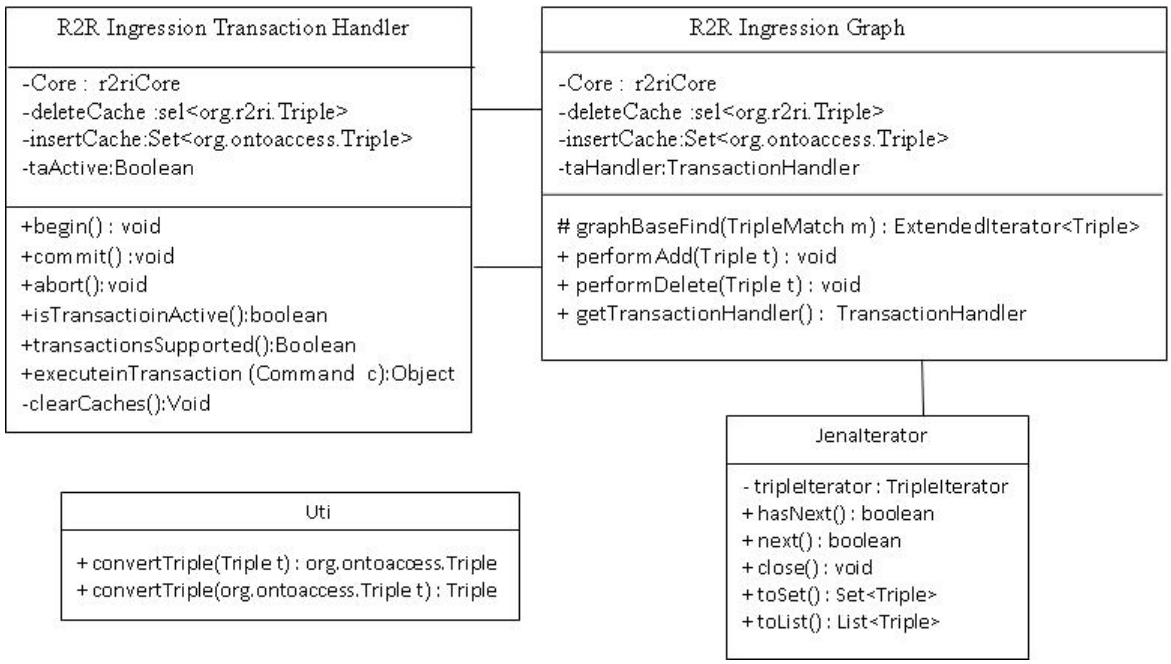

Figure 2. The Jena interface with UML class diagram.

unaware of the knowledge with the storage system of the databases schema. Sometimes it is not all ways necessary for the client having acknowledged with the RDB, if the client needs to work with RDF and their interfaces. This is why the specific reason we have highlighted the parsing feedback protocol to overcome the above said problem. The main intention of this parsing feedback protocol is to identify these invalid write requests pertaining to the relation databases and send back these invalid write requests to the client for further processing.

The parsing feedback protocol in the R2RI is used as a cross-layer feature. The identification of all the invalid requests is probably done during the arrival of incoming request and further these requests are analyzed and processed for translation. The complete process is performed in the update core. All the invalid requests are sorted and finalize these requests as invalid elements. The feedback after the final result is stored with a specific format in the core layer and further exposes these results over the service end point to the *data access interface* layer. The interfaces further process these request issues as per the needs and requirements. For example - the Jena interface on the R2RI cannot transform the request to java exception. The network service endpoint on the other hand converts all the feedbacks to the respective parsing feedback. All the requests are placed in the RDF format and can be retrieved by the client with GET request on the Uniform Resource Locator (URL).

Finally, all the above sections depict the usages and approaches of feedback with respect to our parsing feedback ontology. At the end we have presented an example based on the RDF using the service endpoint.

5.1 The Feedback Types

We have recognized five different causes for the non-validation of write requests during the translation process. All these aroused because of the gap existing during the translation process from RDB-to-RDF. Hence, we have clearly depicted the appropriate types of feedbacks which are described as under.

5.1.1 The Missing Triple Feedback

If a request is made on the data for specific attribute, and that attribute is missing in the data then, immediately a feedback called Missing feedback protocol is generated. Missing feedback accomplishes two cases of feedbacks. Primarily, if data is supposed to be inserted in the table, then a new record is also framed within the database schema, but with at least one missing attribute. Secondly, if data is supposed to be deleted where that data is linked to subset of another record holding at least one attribute. At such situations, in both of the scenarios the values of the attribute are set to zero. One important point is very keen in the scenario that all the feedbacks with Missing triples are aborted.

5.1.2 The Unknown Subject Feedback

If a request is done on RDF Triple that comprises of a subject attribute, but unable to map in the RDF schema, leads to unknown subject feedbacks. Such subjects in RDF cannot be stored. The type of feedback is very specific to insert requests, but is not applicable to non-existence of triples, because no request operation can be made on the triples if at all they are not available. Hence, these can be ignored[11] without any secondary thought. Anyway an unknown subject feedback can be accessed for acknowledgement of non-existence of triples.

5.1.3 The Unknown Triple Predicate Feedback

If a request is done for an RDF triple, that comprises of predicate but unable to map in the RDF schema leads to unknown Triple predicate feedback. Such predicate attribute cannot be stored in the RDF. The type of feedback is very useful for inserting the requests and is completely not applicable for non-existence of triples. No request can be made on triples, if they are absconding. Hence, these can be ignored[11] without any secondary thought. Any way this feedback is accessed for the purpose of acknowledging the non-existence of predicate triple.

5.1.4 The Non Matching Triple Feedback

If a request is done for an RDF table in the mapping process of the database schema, the request process cannot be done because of the existence of already updated record in the database. The request cannot find the respective matching triple in the updated record. Hence, it results an error message and returns a feedback called Non Matching triple feedback. In this type of feedback, the predicate and subject may be ensured for mapping or can be declared either as unknown triple feedback or Unknown subject triple feedback. The feedback is very specific to insert requests, but is not applicable to the non-existence of the triple, because no

request can be made on the triples if their existence is absconding. Hence, these can be ignored[10] without any secondary thought. Any way this feedback is accessed only for the purpose of acknowledging the non-existence of triple.

5.1.5 The Default Triple Added Feedback

A request is done on an RDB but the RDB lacks the attribute for the request made. Since, the attribute is filled with the default value which is already predefined. The default values are supplemented in the new record in the relational schema, if the values are assigned by the respective client. The major activity of the default triple added feedback is to intimate the client about the supplemented information generated during the translation process. There are certainly two issues pertaining to this feedback. Primarily, if new data is inserted in the database scheme, a new record is also created. This new record must at least hold any one attribute with default value. Secondly, if a data is removed from the database schema, at least one attribute's default values must be removed. At such circumstances the actual data might be erased but the data is essentiality restored with the given default values. Anyway, this feedback is accessed for the purpose of acknowledging the triples and no request is terminated under any circumstances.

Multiple feedbacks are often generated by most of the write requests with different or similar types. In the next presentation all the above mentioned five feedbacks are united as parsing feedback ontology called single feedback message system to the user.

5.2 The Parsing Feedback Ontology

All the write requests of the translation process which are in the form of parsing feedback are produced to the client in the form of RDF assisted format. The RDF assisted format is elevated by our parsing feedback ontology which is briefly explained in the context.

Examples:

The following mentioned listing 2 depicts a document pertaining to feedback in standard RDF assisted format. The example listing 2 comprises of three independent instances of feedbacks which are briefly summed up as under.

```
1. @prefix fb: <http://r2ri.org/feedback/> .
2. @prefix rdf: <http://www.w3.org/TR/1999/REC-rdf-syntax-19990222.
3. @prefix rdfs:<http://www.w3.org/2000/01/rdf-schema#> .
4. @prefix xsd: <http://www.w3.org/2001/XMLSchema#> .
5. @prefix dc: <http://purl.org/dc/elements/1.1/> .
6
7. fb:FeedbackMessage.1 a fb:Feed.backMessage;
8. fb:hasFeedback fb:FB1,
9.                fb:FB2,
10.               fb:FB3;
11. dc:date "2016-10-05T13:37:21" .
12
13. fb:FB1 a fb:Missing.Triple;
14. fb:action fb:Abort;
15. fb:level fb:Fatal;
16. fb:source fb:Insert;
17. fb:expectedSubject http://localhost:2010/uuc/Student3002;
18. fb:expectedPredicate http://xmlns.com/foaf/0.1/mbox;
19. fb:expectedObjectDatatype xsd:anyURI;
20. rdfs:label "MissingTriple";
21. rdfs:comment "A mandatory triple is missing ...".
22.
23. fb:FB2 a fb:DefaultTripleAdded, rdf:Statement;
24.     fb:actionfb:Ignore;
25.     fb:level fb:Info;
26.     fb:source fb:Insert;
27.     rdf:subject http://localhost:2010/uuc/Student2026;
28.     rdf:predicate http://r2ri.org/edu#grade;
29.     rdf:object "1";
30.     rdfs:label "DefaultTripleAdded";
31.     rdfs:comment "A default triple was added ..." .
32.
33. fb:FB3 a fb:NonMatchingTriple, rdf:Statement;
34.     fb:actionfb:Abort;
35. fb:levelfb:Error;
36. fb:sourcefb:Insert;
37. rdf:subject http://localhost:2010/uuc/Student2001;
38. rdf:predicate http://xmlns.com/foaf/0.1/firstName;
39. rdf:object "John";
40.     fb:expectedObject "Bob";
41.     rdfs:label "NonMatchingTriple";
42.     rdfs:comment "A triple was detected ..." .
```

Listing 2 Semantic Feedback Example

At lines 1 to 5, the initial phase of the feedback document comes with name space prefixes as serialized by Turtle RDF[11]. At the lines 7 to 11, the major feedback message is described in detail with date and time. At lines 8 to 10, the instances of individual feedback from the major feedback message is described. The remaining section of the document comprises three instances of feedback.

The first feedback instance at line 13, fb:FB1 depicts the existence of Missing triple feedback, as it is highly impossible to process a write request with missing mandatory triples. At line 14, the mandatory missing triples are completely terminated. At Line 15, fb:Fetal explains the cause of the severity of the terminated missing triples. At line 16, it is observed in the feedback that the raised request contains an insert request. Always a missing triple feedback consists of a catalogue describing the missing triples. At lines 17, 18, and 19, the expected subject, expected predicate and the data type of the concerned object from the original requests are gathered from the mapping definitions. The lines 20 and 21, depicts the last phase describing the feedbacks. At line 23, default Triple Added is considered as the second type of feedback. The request is not processed since the attribute is not empty. In fact it is filled with predefined values in the Triple. Hence, at line 24 it is not considered. At line 25 the ignored request is accessed only for information for client. In the lines 27 to 29, the supplemented data which is in the form of Triple representation is facilitated with the feedback with RDF[14]. The line 33 describes the final feedback called non matching Triple feedback request. The Line 34 describes about the terminated request. At line 35, fb:Error explains the Severity about the terminated request. The severity depicts that the request can be processed, but the respective triple cannot be stored. Apart from this, the already existing triple attribute is validated. At lines 37 to 39, the feedback comprises of the already not requested Triple. And at line 40, the objects comprising of triples are stored in the database schema.

It is very clear and the important aspect is to remember that each and every feedback instance has one fbi action property. It denotes that a request is terminated if and only if one of the feedback instances orders this fb:action property to fb:abort.

6. Extensibility

In the present paper, we have showed the simplicity and extensibility of our approach R2RI. We also elevated the R2RI approach where it is extensible in adding more data access interfaces. We narrated the extensible features of R2RI with the example implementations for interfaces using linked data, ChangeSet and Jena.

6.1 The Linked Data Interface

One among the examples for read only access is the linked data interface. The interface is simpler for implementation and about hundred SLOC in Jena Technologies were developed. It enhances enormous support for queries related to linked data type through Service point. The interface uses URI as input and then sends back all the triples as URI subject. For this purpose the interface constructs a new Triple pattern with the provided variables as objects and predicate. The URI is used as the prescribed subject. The newly designed pattern is immediately transferred to the R2RI core for further evaluations and translation process. The output of the translation processes yields a Triple Iterator which is duly merged as HTML Port iterator. The HTML Port Iterator produces the triples in the HTML. This enhances the network service end point to facilitate the result oriented pages to the respective callers.

6.2 The Change Set Interface

One among the examples for the Write accesses data is the ChangeSet interface. The ChangeSet interface is easily accessible by the network service end point for the implementation or development of a protocol called ChangeSet Protocol. The protocol proved itself in the implementation process in generating of about 100 SLOC in Java Technologies. It comprises ChangeSet parser, the actual interface for the protocol implementation. The requests of the ChangeSet are attachable to the ChangeSet ontology. The ontology comprises of a subject and two sets of relevant triples. Out of the two sets of triples, one set of triple is exclusive for removing and the lateral is for adding. Jena frame work is extensively used in our ChangeSet interface for the purpose of implementation. The frame work is used to parse the existing requests and extracts the changing subjects. The same process is also applicable for the rest of the two triple sets. The frame work then transforms the already parsed triples to the triples representation set in our approach R2RI. The complete transaction process is executed in one data database transaction to ensure the atomicity of the ChangeSet request with respect to the removal and addition of the Triples.

The above described content clearly elaborates and explains the simplicity of R2RI and demonstrates the

implementation process of interfaces for data accessing. It states its requirement for the generation of code in the Java for the core layer and estimates an approximate of 800 SLOC to be implemented in Java Technology.

6.3 Jena Interface

The Jena interface in our approach is considered as an example interface that requires both the read/write access to the data. The implementation process of the Jena interface was clearly discussed in section 7. From the performance perspective it is observed that it is one among the various complicated interfaces and hence only 250 SLOC is implemented in java.

7. The Experimental Setup

The complete experiment was carried out on a Note Book of make Dell Inspiron, 3GB of RAM of type DDR2 with 667 MHz, Intel Core 2 dual core Micro Processor with speed of 2.33 GHz, HDD of type SATA with 320 GB and 7200 RPM, Operating System of Windows 8 professional from Microsoft Inc. We used the following tools as software configuration in carrying out the experiment. They are Java Runtime (JRE) V.1.6.0-17 and MySQL V.5.1.4.5 as the data base. MySQL with default settings has been used widely for all the approaches and systems. The nodb_buffer_pool_size was raised to 64MB through the configuration file my_cnf. The complete setup was experimented with heap allocation of 1024 MB space. In our experimental setup, we have also reused, various data sets Berlin SPARQL[2] Bench Mark BSBM in wide variety of sizes ranging from multiples of hundred million triples. The reused datasets generate new data sets from existing BSBM. D2R's mapping was also reused in our approach initiated by the BSBM team. The complete experimental setup comprises of two phases. They are the Query phase and the Update phase.

The task of the query phase is to evaluate the performance issue with respect to the queries of single triple pattern. There are eight different kinds of triple patterns out of which one pattern does not have variable. i.e., a strong triple and the other pattern have only one variable, in a database schema. The translation and query evaluation process is briefly depicted below with almost fifty result triple sets. The update part of the experimental setup evaluates the performance issued with deleting and adding single triple. On the other side, B and C are a group of eight and thirteen triples, comes in to enforcement that affects a table (single) and multiple tables in the relational schema. The achieved results were the average of the Bench marks done repeated after two attempts. All of the systems undergo query part execution called as SUTs (System under Test). The Jena SDB, R2RI and the Jena TDB in the D2R are not validated for updating data, and hence, they are termed as Non-Supportive for data updates in the update part.The following are the variety of releases we came across for the experimental set for SUTs (System under Test) with default settings. They are the R2RI V.O.324, Jena SDB V 1.3.0 with indexed data schema layout, D2R V.0.7 and Jena TDB V.0.8.4. The R2RI layered architecture platform states the demerits pertaining to the performance issue with respect to the approaches of RDB-to-RDF mappings. In our current context we dropped the comparisons with R2RI to Jena SDB, native triple store, an RDB triple stores and D2R. The experiment was carried out on Jena API for deleting, adding and querying the triples. The final report presented that the performance issue was much better on R2RI compared to Jena SDB and D2R.

7.1 Results

The Table 1 describes about the result formed after the experimental setup with query benchmark which are equivalent to multiples of hundred million triples. The foremost column in the result table denotes the size of the data set and the name of the approach being used. The remaining eight columns depict the result times represented in milliseconds for each of the triple set (strong subject, object and predicate). The triple set is denoted as the combinations of three letters S.P.O and a (?) question mark. The question make denotes the variables. For example -if the pattern (S P ?) is existing, then it is read as a triple pattern comprising of strong subject, predicate and variable object.

The results revealed the fact that, the R2RI approach for the triples with known subject is better when compared to D2R, Jena TDB and the Jena SDB. For unknown predicates, the R2RI performs well (comparable) for patterns compared to D2R. In general, R2RI performs well comparable for pattern with D2R. In general R2RI is comparable for the known predicates. The Jena TDB's performance is relatively good for triple patterns with the unknown and known subject and is therefore better than the R2RI. The Jena SDB proves

itself to be better in performance issues compared to the R2RI in the pattern like (? ? o) and (? P O) and is very poor for the remaining two patterns. Suppose, if a data base schema index is generated on a specific attribute, the patterns performance can be gradually improved in the D2R as well as R2RI. Practically, the attribute is being mapped to the concerned property P. various tests have resulted that this approach would reduce the repetitions of the triple with known subject patterns. Jena SDB is the only interface that is capable to evaluate (? ? ?). Jena SDB crashes the large volume of dataset even if the heap memory is tripled to 3072 Mb with java class 'Java.Lang.OutOf Memory Error'. The Table 1 represents the data sets with update Bench mark results equivalent to multiples of hundred million of triples. The foremost column in the table depicts the size of the data set and the approach being used. The rest of the six columns represent the results time, measured in terms of milliseconds for deleting and adding of three triple sets. A, being the first triple, B Comprising Set of 8 triples, that duly affect only a single table in the relational schema. The C comprises of thirteen triples which is enforced in affecting group of tables (multiple tables). Hence, we report our result analysis pertaining to read only access with R2RI, Jena TDB, Jena SDB and D2R for all queries.

The results revealed that the R2RI can be easily utilized for deleting and adding of triples in the datasets irrespective of the size of data and triple set. The performance fluctuates with respect to the deletion of the triples. On the close observation, it is enlightened that Jena SDB transforms the deleted triples to its corresponding SQL statement to execute Join operation over big tables. On the contrary, The R2RI transforms the deleted triples as Join-less SQL statement for each and every affected Table 1. The performance of the R2RI is far better compared to Jena TDB for larger volume of datasets. The performance can be further raised by deleting or adding the total number of triples. Figure 3 shows the experimental result for the query bench mark with 1Million triples. Figure 4 shows the experimental result for the query bench mark with 100Million triples. Figure 5 states the sample screen shot of the result claimed from the query with add and remove for author relational schema is also presented.

8. Cessation

In the present context of our paper, we presented the approach of R2RI that perfectly enhances read/write access to the data in the relational schema. We also depicted various approaches for the data access in semantic web and a

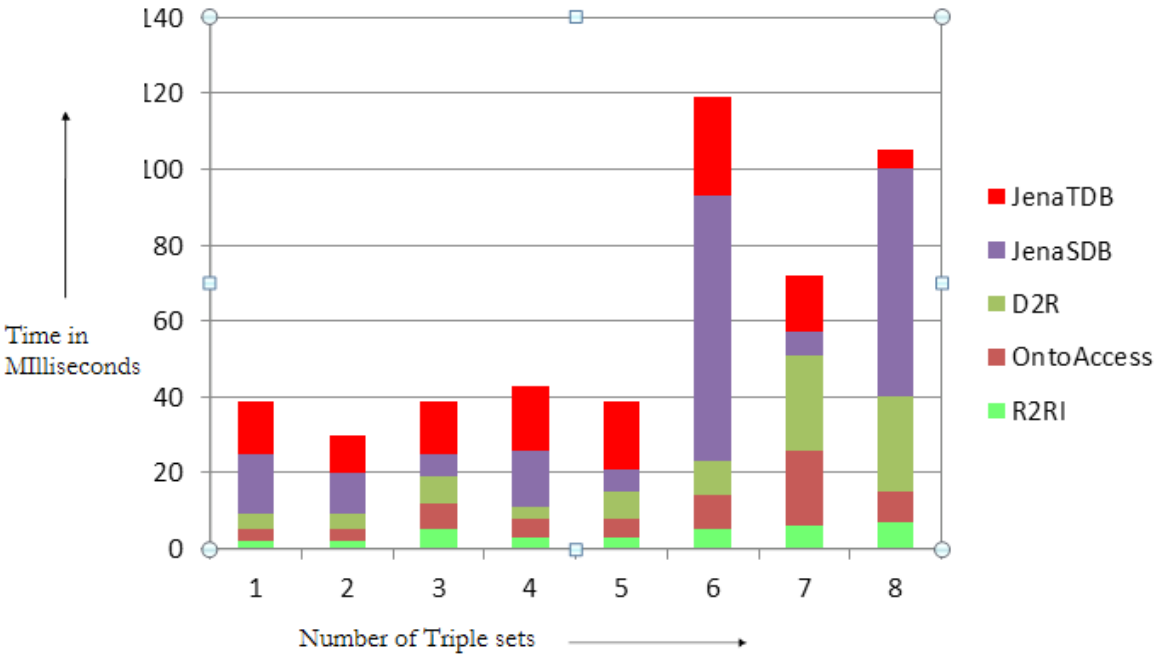

Figure 3. Comparison result graph for 1M triple set.

Table 1. Result sets for various approaches for Query Bench Mark

1 M			R2RI	OntoAccess	D2R	JenaSDB	JenaTDB	10 M			R2RI	OntoAccess	D2R	JenaSDB	JenaTDB	100 M			R2RI	OntoAccess	D2R	JenaSDB	JenaTDB
(subjects	predicates	objects)	2	3	4	16	14	(subjects	predicates	objects)	9	12	15	20	32	(subjects	predicates	objects)	20	30	32	60	87
(subjects	predicates	?)	2	3	4	11	10	(subjects	predicates	?)	10	20	22	12	16	(subjects	predicates	?)	25	30	45	60	78
(subjects	?	objects)	5	7	7	6	14	(subjects	?	objects)	10	15	25	18	20	(subjects	?	objects)	18	20	28	40	60
(subjects	?	?)	3	5	3	15	17	(subjects	?	?)	12	16	18	12	19	(subjects	?	?)	3	4	7	45	40
(?	predicates	objects)	3	5	7	6	18	(?	predicates	objects)	10	40	60	30	40	(?	predicates	objects)	75	99	250	60	316
(?	predicates	?)	5	9	9	70	26	(?	predicates	?)	15	70	78	90	25	(?	predicates	?)	100	123	234	532	200
(?	?	objects)	6	20	25	6	15	(?	?	objects)	25	40	45	30	40	(?	?	objects)	135	195	365	50	99
(?	?	?)	7	8	25	60	5	(?	?	?)	30	37	40	*	20	(?	?	?)	250	513	580	*	300

platform approach is very essential to overcome the several repetition implementations in the translation process of RDB-to-RDF. On our survey, we clearly identified the basic three core operations to be definitely implemented in the data operation. The three core operations indicate the need for querying a triple pattern, need for adding triples and need for removing unwanted triples. We also implemented three operations in our R2RI approach with

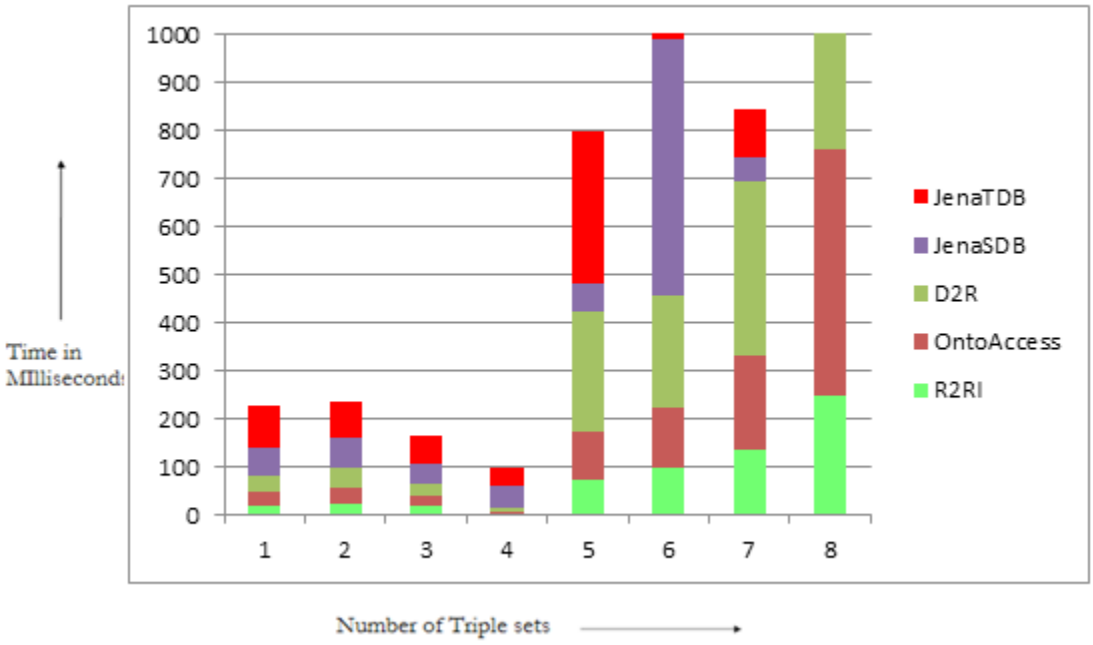

Figure 4. Comparison result graph for 100 M triple set.

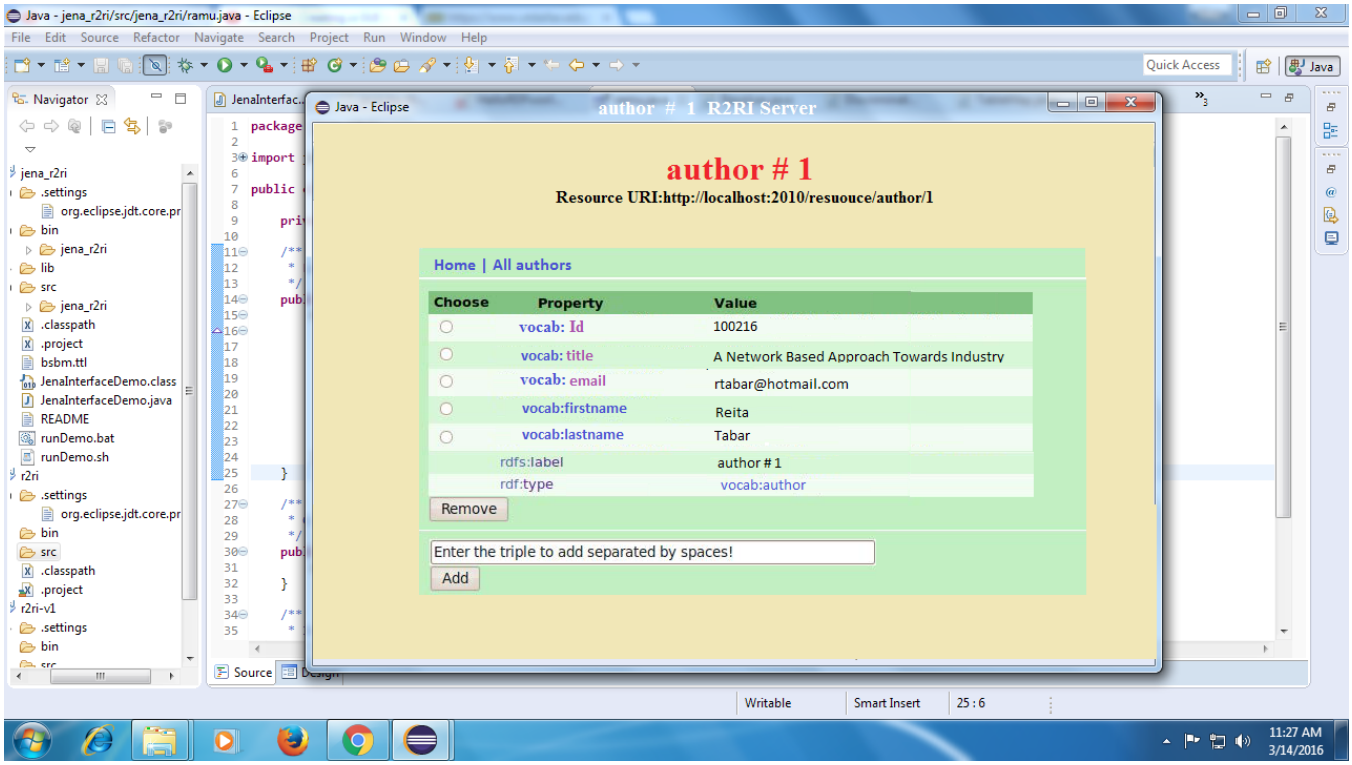

Figure 5. Sample result screen.

describing architectural decisions and developed the accessibility of data using various interfaces through the interface layer. We strengthened parsing feedback protocol to fill the gap between the relational schema and Resource data framework. The parsing feedback protocol acknowledges the client about the invalid write requests in specialized RDF format. It enhances the client to change the request to his convenience. We showed and presented a detailed discussion stating that this approach is very comparable and better to the currently existing RDB triple stores and the read-only RDB-to-RDF mappings. We further introduced the scope for RDB-to-RDF bidirectional approach strategies with proof of implementation. The formal definition introduced by us for our mapping approach R2RI is purely bidirectional and is not influenced by view update problem.

9. References

1. Prud'hommeaux E, Seaborne A. SPARQL query language for RDF, W3C recommendation. 2008. Available from: http://www.w3.org/TR/2008/ REC-rdf-sparql-query-20080115/.
2. Ogbuji C. SPARQL 1.1 graph store HTTP protocol, W3C working draft. Available from: http://www.w3.org/TR/2011/.
3. Ogbuji C. SPARQL 1.1 graph store HTTP protocol, W3C working draft. Available from: http://www.w3.org/TR/2011/ WD-sparql11-http-rdf-update-20110512/, 2011. 12/05/2011.
4. Malhotra A. W3C RDB2RDF incubator group report. 31 Jan 2009. Available from: http://www.w3.org/2005/Incubator/rdb2rdf/XGR-rdb2rdf-20090126/,2009.
5. Barrasa J, Corcho O, Perez AG. R2O, an extensible and semantically based database-to-ontology mapping language. Proceedings of the 2nd Workshop on Semantic Web and Databases. 2004 Aug; 3372:1–17.
6. Malhotra A. W3C RDB2RDF incubator group report. 2009 Jan 01. Available from: http://www.w3.org/2005/Incubator/rdb2rdf/XGR-rdb2rdf-20090126/,2009.
7. Bancilhon F, Spyratos N. Update semantics of relational views. ACM Transactions on Database Systems. 1981; 6(4):557–75.
8. Lee TB. Relational databases on the semantic web. 2009 Jan 25. Available from: http:// www.w3.org/DesignIssues/RDB-RDF.html.
9. Furber C. Ontology-based data quality management: methodology, cost, and benefits. Proceedings of the 6th European Semantic Web Conference Germany; 1981. p. 1–2.
10. Hert M, Reif G, Gall HC. Updating relational data via SPARQL/Update. EDBT Workshop Proceedings; USA: 2009.

11. Erling O, Mikhailov I. RDF support in the virtuoso DBMS. Proceedings of the SABRE Conference on Social Semantic Web. 2007. p. 1–8.
12. Javubar SK, Jaya A. Natural language to SQL generation for semantic knowledge extraction in social web sources. Indian Journal of Science and Technology. 2015 Jan; 8(1). Doi: 10.17485/ijst/2015/v8i1/54123.
13. Schenk S, Gearon P, Passant A. SPARQL 1.1 update, W3C working draft. 2010 Jan 01. Available from: http:// www.w3.org/TR/2010/WD-sparql11-update-20101014/, 2010.
14. Manola Z, Miller E. RDF primer, W3C recommendation. 2004 Feb 10. Available from: http:// www.w3.org/TR/2004/ REC-rdf-primer-20040210/, 2004.