Indian Journals of Science and Technology, Vol 9(20), DOI: 10.17485/ijst/2016/v9i20/92516, May 2016
ISSN (Print) : 0974-6846
ISSN (Online) : 0974-5645

Bridging of Data from Legacy Software to new Software using R2R Ingression

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

[1]Department of Computer Science and Engineering, SBIT, Khammam - 507002, Telangana, India; vsitaramu.1234@gmail.com
[2]Department of CSE, VNRVJIET, Hyderabad - 500090, Telangana, India; raveendrababu.b@vnrvjiet.in

Abstract

Background/Objectives: Software engineering is a state of art providing enormous solutions to the clients. Software engineering methodologies keeps on changing from day to day. This changing nature of software gave place for development of new languages, technologies, new tools in the software applications and many frame works that gained speed in replacing older systems. Hence, the main objective in presenting the paper defines in "bridging" the older software to the latest version new technology software. This enables data to meet accurate requirements in data mining from the data bases. **Methods/Statistical Analysis:** In the storage of databases, the relational databases are used utmost for storing data in very large amount which should be with good and high integrated security to safe guard the data semantics. In order to support the objective, we have introduced an R2R Ingression approach – a methodology to bridge old systems to new technology software in RDB and RDF with various mappings and architectural patterns that duly define the bridging process for RDB and RDF. **Findings:** A clear and contrast elaboration have been extracted between the two distinguished example softwares as presented in our context. We stated the relations of old systems and new systems softwares with the example mappings using R2RI, so that legacy softwares can be easily transparent to the new technology softwares. This kind of bridging is so essential to enhance the usage of old legacy systems at the current latest technologies. A Semi automation tool is apprehended and transformed to a complete automation tool to give additional benefits to the user as well as the administrator. **Applications/Improvements:** This paper covers the development of a new methodology called R2RI which is an RDB-to-RDF mapping with integration approach that contain architecture and mapping which enables the RDF-based Read/Write access to enrich the various characteristics of RDB. So the R2R ingression mapping helps to align the RDB and RDF.

Keywords: Relational Database, Resource Description Framework

1. Introduction

The changing nature of software gave place for development of new languages, technologies, new tools in the software applications and many frame works that gained speed in replacing older systems. The scope of the paper represent RDB-to-RDF platform about its facilities in translating the data from the legacy systems to the new generation semantic web applications. A semantic layer is created by the R2RI and is placed on the top layer of the relational schema. The semantic layer is rich in facilitating services like read/write access to the existing relational schema. The mediator between the RDF and the relational schema translates all the requests of semantic web[1] to the corresponding SQL. This enriches the data co-ordination between the RDF and relational Schema. It also exploits the merits of advanced database technology with respect to security, scalability, querying performance and the transaction support.

*Author for correspondence

The current paper focuses on a case study for R2RI and clearly mentioned how R2RI is properly utilized to navigate with advanced Eclipse-based software evolution framework EVOLIZER[2] to SOFAS[3]. SOFAS is generally a collaborative, distributed and service oriented software analysis platform. We, in the case study initiated the need and importance of the usage of use cases in the current RDB-to-RDF approaches and promoted utility of use case in the R2RI.

The structural organization of the paper is organized as follows. The Sections 2 and 3 entails the introduction for software evolution frame work EVOLIZER and collaborative software analysis platform SOFAS. The bridge between the two platforms is R2RI. Hence, in Section 4 R2RI fills the gap between the RDF and the relational model. A detailed description pertaining to the architecture of a R2RI and various mapping principles are briefly introduced. The Section 5 explains how we successfully utilized the approach R2RI in advancing the software frame work EVOLIZER to software analysis SOFAS in the form of a detailed case study. The Section 6 concludes with a conclusion.

2. The EVOLIZER

In the recent times a research platform for software evolution analysis was generated and named it as EVOLIZER. It is integrated with Eclipse IDE for better analysis of the software.The idea behind the development of EVOLIZER is based on (Released History Database) RHDB[4]. It was developed in such a way that it comprises group of Eclipse plugins to satisfy the needs of integrated data gathered from different locations of software database repositories. The various database repositories include mailing lists, tracking of issues and version control. This enriched the software evolution system and uncovered the different facts of the data. The example considered is refinement of a software program making it free from bugs and errors. It is all to increase the levels of quality in the software program especially in the source code model.

EVOLIZER is a known legacy system. The usage of legacy system in the current generation is still under process. EVOLIZER being a legacy system, service its services to the present market. It is quite complex for the legacy systems to get upgraded with latest advances in the current technology. EVOLIZER when integrated as a tool in the eclipse cannot cope with the latest advances and algorithms with current technology Therefore, it is very difficult task to integrate and interlink information from RHDB to other data sources. One among the causes for this is that the information residing in the RHDB is limited to non-local and all the entities in the legacy system (RHDB) lack universal Unique Resource Identifiers (URI'S). Hence, all the legacy systems are standalone pertaining to their data. The data in the legacy systems cannot be exploited on the web. Some of the models in the EVOLIZER lack with the following semantics:

- Transitivity.
- Symmetry.
- Cardinality etc.

It is always advisable to the get the data from EVOLIZER and RHDB to the corresponding semantic web to make use of data to the maximum extent. If a bridge is levied between the legacy systems to an advanced technology; it becomes easier and faster means to access data.Though legacy system EVOLIZER is an old system, its services are still under usage with RHDB and consists of hundreds of software life cycle systems. Importing this large volume of data is a bit of typical task because data is to be imported and navigated across bug tracking and version control systems. Obviously, it may consume months together for completing the task. Sometimes data repositories may also not be available over the web resources.

In order to overcome the above said limitation, it is very essential to establish migration path from the legacy system EVOLIZER to the current generation software evolution platforms so as to make use of old data wisely in the coming years too.

3. SOFAS

The software evolution framework EVOLIZER has given a chance to analyze and design software with many features. The design of good software must encompass easy navigation, no language limitation, easily adoptable by the client and not necessary to configure many tools manually. Based on the design concept of good software, we raised the "Analysis of any software service"[5]. It states that it becomes easy to access large amount of information through various tools available from the web resources. We gradually developed this concept to a flexible platform called light weight process "Software Analysis Services"[6].

Describes[7] about the REST architecture and the same principles are followed by SOFAS. The principles in the 'Representational State Transfer' allow efficient use of software on the web. The REST architecture comprises the following three major parts:

- Software Analysis web services.
- Software Analysis Broker.
- Software Analysis ontologies.

All the services and functionalities by the REST architecture to the data are done by means of REST interfaces. The 'Service Manager' in the architecture is also called as software analysis "Broker" facilitates the interfaces between the user and services. The taxonomy of software analysis comprises bunch of registered services. All the services featured by the architecture are briefly described in the form of semantic representation and thus users browse and surf for the required content they need. SEON (cf. section 5.2) is determined as the ontology and is used to represent the amount of data existing in the RDF and services offered by the RDF to produce the data.

4. R2RI as Bridge to the New Software Analysis

R2RI acts as a bridge between the semantic web application and the relational data. R2RI as a mediation platform enables to operate the semantic applications over the relational schema. R2RI facilitates the semantic layer which is placed on the top of the relational schema. It specifically enhances read/write access to the data. All the semantic web requests such as update requests, write requests, and read requests are translated to corresponding SQL statements. Therefore, R2RI deteriorates the usage of synchronizing and mirroring the information in the relational data. The RDF representation and the relational data models do operate on the same data. This scenario resulted in incorporating co-operative method to access data by both RDF and relational applications. Besides this, the mediation platform R2RI further deploys the merits of the current latest technologies with respect to security, scalability, transaction support and performance. All the existing approaches today are read only and can be analyzed and queried for use but cannot be modified. The R2RI on the other hand facilitates the RDF to read/write the data. It also enhanced the transitions migrating from old legacy systems to the advanced web-applications.

4.1 R2R Ingression Architecture

The R2RI is designed and developed with an intention to extend the platform. The R2RI is enriched with encapsulation technique that hides the translation logic during the conversion from RDB-to-RDF in the interface layer. The core layer in the RDB-to-RDF converts all the RDF requests to its corresponding SQL queries by interacting with database schema. Several approaches are put in to action to expel the wide variety functionalities of R2RI to the applications of RDF. It is capable in translating all the interface related operations to the extensible platform R2RI. The already translated interface operations are related to the interface layer in "interface-specific format". This enraged the design and development of many data access interfaces for the translation process of RDB-to-RDF in the core layer. At, present, the R2RI implementation approach makes use of SPARQL[8], Linked data[9,10], SPARQL/Update[11] and other semantic web frame works such as RDF2Go Sesame and Jena for data access. Most of the data access interfaces are accepted and addressable by means of any standalone server called HTTP Service end point. The fact of the R2RI is that it can be easily associated with many applications in the RDB. The data can be accessed from these applications by the data access interface with the help of API's. The Figure 1 depicts the architectural overview of R2RI and clearly presents how R2RI can be bridged between the legacy System EVOLIAER RHDB to the latest software analysis SOFAS. Further in the Section 4.2 we clearly stated the principles of mapping for the R2RI.

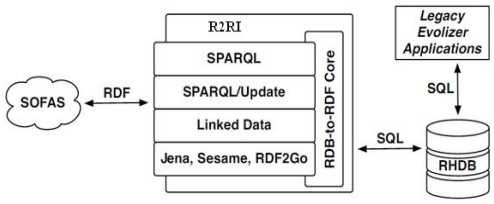

Figure 1. R2R ingression architecture.

4.2 Extended mapping principles for R2RI

R2RI mediation approach is intended to map the concepts from the relational schema to the corresponding ontology terms. We initiated the R2RI for the approach ONTOACCESS which is a bidirectional mapping from RDB-to-RDF and vice-versa. It greatly supports RDF based read/write access to the data existing in the relational schema. All the existing languages today are completely dependent on read access only to the data. The use cases pertaining to the read only data access are obviously not relevant for write access as explained[12] in D2R/Update. The attributes existing in the database schema are mapped corresponding to the properties in the ontology and tables are mapped to classes. Link Table in the relational model has a special importance. N:M relationships are represented by the link tables in the schema. The helping constructs are not required by the RDF because link tables are directly mapped to the corresponding properties rather than using classes. The R2RI holds necessary information such as integrity constraints as well as various data types of the relational model. The outcome of the above integrity constraints generates a mapping language which is not high expressive to the existing mapping languages. The highly normalized relational databases are mapped with the R2RI. One such example is Hibernate.

The Listing 1 in our presentation depicts the mapping constructs examples of R2RI. The examples comprised three important constructs. The examples include the namespaces which are describes as under: R2RI describes the vocabulary of our mapping language. Here *ex* is represented as a namespace. Ver. depicts the version control of respective ontology.

```
1 a)  ex : revision a                r2ri : TableMap;
2     r2ri :hasTableName             "Revision";
3     r2ri : maps To Class           ver:Version;
4     r2ri :uriPattern
      "http://.../revision_%%number%%";
5     r2ri : has Attribute           ex:revision_number,
6
7 b)  ex :revision_number    a       r2ri : AttributeMap;
8     r2ri :hasAttributeName                "number";
9     r2ri :mapsToObjectProperty            ver:hasID;
10    r2ri : dbType                [ a r2ri: VarChar;
11                                   r2ri : length 255 ];
12    r2ri :hasConstraint          [ a r2ri:NotNull ].
13
14 c) ex: release_revision    a     r2ri :LinkTableMap;
15    r2ri :hasTableName             "Release_Revision";
16    r2ri :mapsToObjectProperty     ver:comprises;
17    r2ri :hasSubjectAttribute      ex:rr_release;
18    r2ri : hasObjectAttribute      ex:rr_revision.
```

Listing 1 Example R2RI Mappings

The Listing 1a). In our example clearly states about Table Map describing the mappings, where the table in the data base is mapped to its corresponding class in the respective ontology. At line 2, the Table Map comprises the name of table. At line 3, it describes about the class it is mapped. At line 4, the R2RI pattern comes in the scenario and generates the specific instances to URI's. The double percentage signs represents the attributes of the table (%% number %%). Here the number which is in between the double percentage signs is the primary key and referred as unique attribute. At Line 5, the Table Map represents the attribute maps.

The Listing 1b) in our example clearly states about the respective attribute to the corresponding property in the ontology. At Line 8, the attribute map shows the name of the respective attribute existing in the relational schema. At Line 9, it describes about the property it is mapped. At Lines 10 and 11, it describes the type of the data type used by the database attribute. At line 12, it stated about the constraints being defined on the attribute. For example: Not Null Constraint. R2RI is rich in supporting constraints like foreign key, primary key, default and check etc. R2RI: foreign key, R2RI: primary key, R2RI: default and R2RI: check are widely used constraints across the mapping.

The Listing 1c) in our example represents the link table map. The link table is mapped to the corresponding ontology property. At Line 15, it represents the name given to the Link table. At Line 16, it represents the property being mapped. The N:M relationship in link table is formed by the two foreign key attributes in the relational schema. At lines 17 and 18, the attributes formed through N:M relationship is granted as Attributes Maps. These enhance the following references:

- Provision of name to the attributes.
- Referencing the Foreign Keys to the attribute table.
- Representing the relationships with direction (either from subject to object or object to subject).

5. Overview of Case Study with Software Analysis

In order to strengthen our case study the usage of the use cases are deployed between SOFAS and EVOLIZER. The necessity of using the use cases is to promote the data exchange in the bidirectional way. Primarily, Evolizer comprises large volume of data from the existing software systems. The involvement of use cases fulfilled the need for data exchange between SOFAS and EVOLIZER. The SOFAS, on the other hand must be to be able to access the complete set of data from the EVOLIZER. For this purpose, the SOFAS require read access based RDF to read and import data from EVOLIZER. Secondly, the EVOLIZER imports the historical data and source code from centralized control systems such as SVN and CVS by implementing importers. As technology is gradually increasing, the decentralized systems have gained much demand by the users. Few such examples relating to decentralized systems are Mercurial and Git. Hence, the reason need for importers in SOFAS platform have become very essential. The importers generate data and the generated data is produced to the RDF. On the basis of the ontologies called SEON which is briefly discussed in the Section 5.2, the duly obtained data is highly variable to EVOLIZER because all the currently existing features and tools use this data for leverage. Write access is very mandatory to the EVOLIZER. SOFAS is an advanced software analysis that facilitates software metrics to be computed on the machine for the data. It generates an extensible framework that richly supports many tools to work on the data. After the metric analysis, the data is again expelled to the RDF with only matching relations in the schema of EVOLIZER. The write access with respect to the RDF must be facilitated to the RHDB to avail data metrics to EVOLIZER. The use cases duly employed for migration of data from EVOLIZER to SOFAS states that bridging is essential with read/write access to relational schema. All the approaches till today tend to use only read only access and we developed R2RI that in richly facilitate both read/write access efficiently in a bidirectional way. At the end of this section, we present ontology and relational data models called SOFAS and EVOLIZER and the establishment of bridge between these two modules to fill the gap during transitions. At later stages challenges are also discussed.

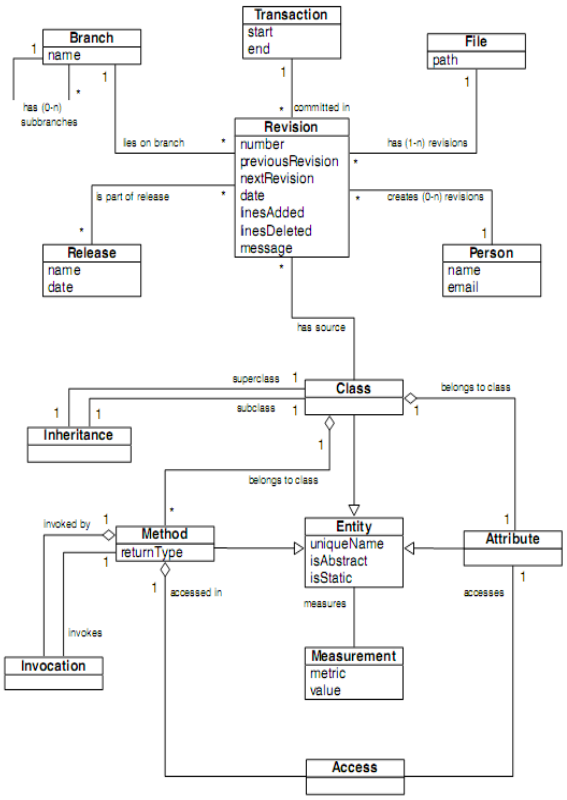

Figure 2. Historical analysis for source code with EVOLIZER data schema.

5.1 Analysis of Software in EVOLIZER with Data Schema

The data model of the EVOLIZER comprises of several aspects relating to the domain of software engineering. In our case study, we concentrated on the historical aspects and the source code of the information. The historical aspects considered in our approach are quite generic as they are applicable for centralized distributed systems and the centralized systems. The following few points depict the importance of our schema. The Figure 2 entails the complete structure of the schema.

One among the entities in the considered schema is revision which is a part of file. In short, a software practitioner appends a specific file and fixes the updates to the version control system. A catalogue maintains track record of the appended files regarding the date, time, type of the context appended and the necessity for appending a file. The release is another entity whichplays vital role in the software lifecycle system. It is referred to as a codename and the versions associated with the release of these files form software. The branch on the other hand features experiments, error control, fixing of bugs etc. After the bug fixation, new code without errors is called bug free software which is associated back to the track. In order to develop consistent software, the source code should be accurate in terms of semantic approach and the systems under analysis must be precise. The newly developed software can be guaranteed only for the release and updates can be modeled for next release with selecting suitable software model. All the classes in the model comprises group of members namely methods and attributes. Therefore, the classes, methods and all the attributes are set as entitles. The entities are referenced and related with various relationships. Inheritance is used across the class to elaborate the structure of class, accesses relationship is used between methods and attributes, and Invocation relationships are deployed between method and another method. All the relationships are considered to be link table or an association class. These relationships are typical for all the real entities. But these are used for the sake of querying in the relational mode.All the entities in the model can be measured by means of software 'Metric' analysis. For example the SLOC written for a class and the "total number of attempts" made for a specific attribute.

5.2 Software Analysis of SOFAS within Ontologies

SEON which is abbreviated as "software Evolution ontologies" describes about the data used by the SOFAS. SEON describes about the various evolutions of software like tracking various aspects of issues, structure of some code various design metrics of software, change coupling and version control etc. SEON is structured as ontology pyramids. We developed and defined high structured ontologies for the major sub domains with similar concepts. On the other hand, we also developed strong low level ontologies for language independent and system specific concepts. The examples for high and low level ontologies include SVN, Git and CVS. In the current chapter, the discussion is limited to the version control and source code terms in the ontology. The Table 1 in our presentation depicts the usage of main classes and their properties pertaining to the source code ontology. Local variables, interfaces and all other exceptions cover the complete domain of ontology.

The Table 2 in our presentation depicts the usage of main clauses and their properties pertaining to the SEON version control ontology.

5.3 R2R Ingression as a Leveling Contract to the New Software

R2RI fill the gap between the Semantic web SOFAS and the RDB schema EVOLIZER by establishing bridge to transfer data. R2RI uses the RDB-to-RDF mapping with enhancing read/write request to RHDB EVOLIZER. The Tables 3 and 4 depicts the sematic mapping representation. We primarily lay emphasis on the EVOLIZER RHDB in the mapping for our case study. The mapping comprises the name spaces like Java.owl, top.owl and versions.owl in the ontologies.

Further, Table 3 depicts all their respective attributes and the domain concept duly mapped from Table 2. The table comprises of four different columns. The first column specifies the name of the table as clearly shown in Table 2. The Second column in the Table specifies the class it is being mapped. The third column in the table comprises the attributes. The fourth column depicts the attributes being mapped to its corresponding properties. A dash in the tables at columns two and four states non-occurrence's of mapping. All the attributes in the table are mapped to its corresponding properties and class is not mapped. All the attributes in the table are mapped to its corresponding properties and class is not mapped

Table 1. Overview of ontology source code

Class: Class	**Class: Method**
→ declaresMethod : Method	→ accessesField : Field
→ declaresField : Field	→ hasParameter : Parameter
→ isReturnTypeOf : Method	→ invokesMethod : Method
→ isSubclassOf : Class	→ hasReturnClass : Class
→ isSuperclassOf : Class	→ isInvokedByMethod : Method
→ hasName : xsd:string	→ isMethodOf : Class
Class: Field	→ hasName : xsd:string
→ isDeclaredFieldOf : Class	**Class: Parameter**
→ isAccessedByMethod : Method	→ isParameterOf : Method
→ hasName : xsd:string	→ hasName : xsd:string

Table 2. Overview of version control

Class: Version	**Class: ChangeSet**
→ hasID : xsd:string	→ hasCommitDate : xsd:date
→ follows : Version	**Class: Branch**
→ precedes : Version	→ hasTag : xsd:string
→ hasCreationDate : xsd:date	**Class: Release**
→ linesAdded : xsd:int	→ hasReleaseDate : xsd:string
→ linesDeleted : xsd:int	→ hasTag : xsd:string
→ hasMessage : xsd:string	

by the table entity in the ontology. Entity comprises of many several concepts and hence it is not mapped. Here, in our case study, we represented entities that comprise sub concepts.

The Table 4 depicts the formation of M:N relationships from the mapping of link tables. The help constructs are not necessary during the formation of M:N relationships in the RDF and hence, all the tables are mapped to the corresponding properties. The table comprises three independent columns. The first and foremost column denotes the name of the link table. From Figure 2, connecting various concepts with a line and is represented as underscore. The second column in the table specifics the property of each and every table mapped. The third column specifies the inverse property.

5.4 Detailed Description

The case study in our presentation clearly states how we efficiently made use of R2RI in establishing bridge to the new technology software. It depicts the migration of data from Legacy Systems (EVOLIZER) to the new technology software analysis (SOFAS). We further demonstrated that RDB-to-RDF mappings are not relevant for this kind of approach because these mappings hold well for read only data. During the reign of our case study, we have encountered many challenging issues related to R2RI. The following below mentioned are the two major issues with probable solutions.

The first and foremost challenge relates with the concept of inheritance in relational schema. The object-oriented concepts greatly depend on inheritance. It plays vital role in object oriented programming. The same inheritance is used by EVOLIZER. The object oriented and the object relational databases do not support the feature of inheritance. Therefore to make use of inheritance widely, three strategic principles are advised in the relational schema[13]. The foremost strategic principle in the relational schema 'table-per-hierarchy' states that all classes containing the property of inheritance are added to a single table. This table further comprises of several

Table 3. Overview of mapping - Part I

table	→	class	attribute	→	property
Revision	→	ver:Version	number	→	ver:hasID
			previousRevision	→	ver:follows
			nextRevision	→	ver:precedes
			date	→	ver:hasCreationDate
			linesAdded	→	ver:linesAdded
			linesDeleted	→	ver:linesDeleted
			message	→	ver:hasMessage
Transaction	→	ver:ChangeSet	start	→	-
			end	→	ver:hasCommitDate
Branch	→	ver:Branch	name	→	ver:hasTag
File	→	top:File	path	→	top:filePath
Release	→	ver:Release	name	→	ver:hasTag
			date	→	ver:hasReleaseDate
Person	→	foaf:Person	name	→	foaf:name
			email	→	foaf:mbox
Entity	→	-	isAbstract	→	java:isAbstract
			isStatic	→	java:isStatic
Class	→	java:Class			
Method	→	java:Method	returnType	→	java:hasReturnType
Attribute	→	java:Field			
Measurement	→	met:Metric	metric	→	met:hasName
			value	→	met:hasValue

Table 4. Overview of mapping - Part II

link table	→ property	inverse property
Release_Revision	→ ver:comprises	ver:appearsIn
Branch_Revision	→ ver:comprises	ver:isOn
Transaction_Revision	→ ver:comprises	ver:commitedIn
File_Revision	→ ver:hasVersion	ver:belongsTo
Person_Revision	→ -	ver:committedBy
Class_Revision	→ ver:hasSource	-
Method_Class	→ java:isDeclaredMethodOf	java:declaresMethod
Attribute_Class	→ java:isDeclaredFieldOf	java:declaresField
Measurement_Entity	→ met:isMetricOf	met:hasMetric
Inheritance	→ java:hasSubClass	java:hasSuperClass
Invocation	→ java:invokesMethod	java:isInvokedByMethod
Access	→ java:accessField	java:isAccessedByMethod

columns. One among the columns named discriminator describes the attributes pertaining to the classes defined in the database. The discriminator column also holds the instances for each and every class.

The second strategic principle in the relational schema table–per-concrete-class defines each and every class from schema in its own table. These tables comprise the attributes in the columns. The super classes to the classes are arranged in the hierarchical order. Therefore, the attributes belonging to the super classes are replicated and stored in the sub classes in our mapping.

The third strategic principle in the relational schema table-per-subclass describes all the available classes in the table. The attributes belonging from the super classes in the table-per-concrete-class are not replicated as columns

in sub classes. In fact, we used a primary key to connect the classes.

Different techniques and approaches are followed by EVOLIZER for different inheritance hierarchies. As an example, we used entity concept for implementing table-per-hierarchy. The table–per-concrete–class is an exclusive strategy because it contains one class per table and the corresponding tables do not interfere with each other. The rest of the two strategies need support to represent the parent-child relationships by using the discriminator column for the mapping. This has become one of the limitations. We resolved this limitation by introducing mapping constructs in to our R2RI approach. Initially, the discriminator columns are supplemented to the strategy table-per-hierarchy. It is easy to facilitate multiple mappings to the tables with inheritance hierarchies by the existing classes because the table already comprise of subset of class.

The Listing 2a) represents a table with discriminator columns and is referred as a concrete mapping by using the constructs to relate each table. This facilitated the support for parent and child relationship in the strategy table-per-subclass. The parent table has the ability and feature to reference another table. This raised the R2RI to identify an issue that multiple tables are created from the application domain in the relational. The newly formed multiple tables can be merged by using primary key.

```
1 a)  ex : method      a          r2ri :    TableMap;
2     r2ri :hasTableName     "Entity";
3     r2ri :mapsToClass          java:Method;
4     r2ri :hasDiscriminator ex:method_type;
5     r2ri :uriPattern       "http://.../method_%%id%%";
6     r2ri :hasAttribute         ex:method_type, ....
7      ex    : method_type a    r2ri : AttributeMap;
8     r2ri :hasAttributeName     "ctype";
9     r2ri :hasValue             "Method".
10
11 b) ex: Method  a  r2ri :     TableMap;
12    r2ri :hasTableName "Method";
13    r2ri :mapsToClass          java:Method;
14       r2ri : hasParentTable      ex:entity;
15       r2ri : uriPattern    http://.../method_%%id%%»;
16            r2ri :hasAttribute     ex:method_returnType.
17    ex : entity  a     r2ri:TableMap;
18    r2ri :hasTableName     "Entity";
19    r2ri :hasAttribute         ex : entity_uniqueName, .
```

Listing 2 R2RI Extended Example Mappings.

The Listing 2b) references a parent table as an example for concrete mapping. The above specified two extensions enriched the R2RI for mapping the inheritance hierarchies.

The challenging issue is concerned with the definitions of RDB-to-RDF mappings. The R2RI mappings are simulated to RDF that enhances easy processing by the computing machines. The users do not have accessibility for this. Defining such maps through manual process is quiet ridiculous because it involves many repetition steps that consume lot of time and may sometime prone to risks in terms of error. Therefore, is very essential to deploy tool to analyze the steps during the translation process for large typical applications those constitute enormous tables. Hence, we designed a tool[14] which simplifies the mappings of R2RI. The tool automatically uses two steps for our mapping from relational schema. Primarily, it implements a prototype mapping with the data collected from the catalogue of database. The ontology terms are also framed and generated from the data sets in the schema. Secondly, the tool facilitates a graphical editor to edit the mappings manually. All the generated terms are then substituted with the actual terms form the ontology. It enhances the editing feature manually to fix the bugs and errors being formed and helps in validating the mappings. The tool is a supportive plugin in the editor of ontology that aims in fast accessing for the terms in ontology.

6. Conclusion

In the current context, the concepts of R2RI which is a RDB-to-RDF read/write mediation platform that facilitates transactions of data from legacy systems to latest systems semantic applications are extracted in detail. R2RI also generates a semantic layer that is duly placed on the top layer of the relational schema. The queries pertaining to update request and semantic related requests are converted to SQL in data base schema. R2RI explicitly presents its advantages in security, scalability and transaction support including query performance.

Our case study presented the success of R2RI in migrating or exchange of data from the advanced systems SOFAS to the legacy systems EVOLIZER. The complete scenario is precisely evaluated with the example mappings. In short, the experiences we faced in our case study

reveals that R2RI is an essential tool for read/write access and facilitates several services over semantic web technology.

7. References

1. Manickasankari N, Arivazhagan D, Vennila G. Ontology based semantic web technologies in E-learning environment using Protege. Indian Journal of Science and Technology. 2014 Oct; 7(S6):64–7. Doi no: 10.17485/ijst/2014/v7iS6/54591.
2. Gall HC, Fluri B, Pinzger M. Change analysis with evolizer and change distiller. IEEE Software. 2009 Jan-Feb; 26(1):26–33.
3. Ghezzi G, Gall HC. SOFAS: A Lightweight Architecture for Software Analysis as a Service. Proceedings of the 9th Working IEEE/IFIP Conference on Software Architecture; Boulder, CO. 2011 Jun 20-24. p. 93–102.
4. Fischer M. Pinzger M, Gall H. Populating a release history database from version control and bug tracking systems. Proceedings Int'l Conference on Software Maintenance, ICSM'03; 2003 Sep 22-26. p. 23–32.
5. Ghezzi G, Gall HC. Towards software analysis as a service. Proceedings of the 4th International ERCIM Workshop on Software Evolution and Evolvability; L'Aquita. 2008 Sep 15-16. p. 1–10.
6. Ghezzi G, Gall HC. SOFAS: A Lightweight Architecture for Software Analysis as a Service. Proceedingsof the 9th Working IEEE/IFIP Conference on Software Architecture; Boulder, CO. 2011 Jun 20-24. p. 93–102,
7. Fielding RT. Architectural Styles and the Design of Network-based Software Architectures. [PhD thesis]. Irvine: University of California; 2000.
8. 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/
9. Berners-Lee, T. (2009a). Linked Data. 2015. Available from: http://www.w3.org/DesignIssues/LinkedData.html
10. Khusro S, Jabeen F, Mashwani SR, Alam I. Linked Open Data: Towards the Realization of Semantic Web - A review. Indian Journal of Science and Technology. 2014 Jun; 7(6):745–64. Doi no: 10.17485/ijst/2014/v7i6/38378.
11. Seaborne A, Manjunath G, Bizer C, Breslin J, Das S, Davis I, Harris S, Idehen K, Corby O, Kjernsmo K, Nowack B. SPARQL Update – A Language for Updating RDFGraphs. W3C Member Submission. 2008. Available from: http://www.w3.org/Submission/2008/SUBM-SPARQL-Update-20080715/
12. Garrote A, Garcia MNM. RESTful Writable APIs for the Web of Linked Data using Relational Storage Solutions. Proceedings of the WWW2011 Workshop on Linked Data on the Web. 2011. p. 1–9.
13. Garcia-Molina H, Ullman JD, Widom J. Database Systems: The Complete Book. Prentice Hall Press; 2008.
14. Brugger N. RDB-RDF mapping generation from relational database schemata. [Master's thesis]. University of Zurich; 2009.