Algorithms for Mapping RDB Schema to RDF for Facilitating Access to Deep Web

Wondu Y. Mallede, Farhi Marir, Vassil T. Vassilev
Knowledge Management Research Centre,
School of Computing, London Metropolitan University, London, U.K
wym0001@my.londonmet.ac.uk, f.marir@londonmet.ac.uk, v.vassilev@londonmet.ac.uk

***Abstract*— Semantic Web sets the standard for a universal and interoperable data representation that is not only readable to the naked eye but also to computers. The use of Uniform Resource Identifier (URI) and the capability to use description logic through semantic ontology languages makes semantic web the favoured framework to represent knowledge. Semantic Web standards play a vital role in making the existing relational database, which is locked behind in the "deep web", available for computer processing. In order to map the relational database in its entirety, the methodology should not only map the data but also the domain specific knowledge. The algorithms and their implementation presented in this paper use the meta-data from the data dictionary to construct the initial semantic repository, while using domain specific knowledge during in-processing stage.**

Keywords-Relational Database Mapping; Domain Specific Knowledge; Semantic Web; Data Mapping Algorithm

I. INTRODUCTION

The current web experience gives us a fairly abundant data. Using a few keywords and common search engines, it usually does not fail to return search results as well. With all its openness, the web gives anyone a chance to contribute ideas to be shared by the whole world about any topic. The web often feels like it is a mile wide, but an inch deep. How can we build a more integrated, consistent, deep web experience? [1].

The ANSI-SPARC (American National Standards Institute, Standards Planning And Requirements Committee) architecture for databases dictates the separation of the conceptual level from both external view (users) and physical level (files). The conceptual level consists of all the entities, their relationships and constraints that channel the data back and forth between the external and physical level. Mapping the conceptual level to semantic Resource Description Format (RDF) [12] makes the "deep web" re-surface for semantic interpretation and processing. The Semantic Framework helps narrow the gap between the "deep web" and the "surface" web.

Semantic Web is a framework which allows information to be represented not only structurally using suitable structural definitions ("data schema"), specific instances of them ("data") and their use ("access rights"), but also semantically using a logical model which allows formal interpretation and sound logical inference about the information ("knowledge"). Relational data models on the other hand lack the capability to represent "knowledge" in spite of their popularity. Since most KR (Knowledge Representation) mechanisms and the Relational Data Model are based on symbolic languages, the ability to represent and utilize knowledge that is imprecise, uncertain, partially true and approximate is lacking, at least in the base/standard models [9]. This lack of capability to represent and process knowledge while it is still in relational model is one of the challenges in Knowledge Management. This research focuses on the development and evaluation of algorithms to map the Relational Database (RDB) schemas to RDF in Semantic Web to allow access to deep web applications. The evaluation and validation of the developed mapping algorithms has been undertaken on a space project management domain-specific application.

Space program projects involve activities like observation, human space exploration, space launch and navigation, operational maintenance, etc. The range of terminologies, standards, unit measurements, and definitions will all be referring to the space domain ontology. The practical evaluation of the developed mapping algorithms has been undertaken on Oracle database system representing the space database schema which is composed of six major relational concepts- Documents, Risks, Non Conformances, Reviews, Actions, and Projects.

Semantic Web has a data model as part of its architecture that will be used as a repository to store 'semantic data'- RDF. RDF is a format to store data in Semantic Web and will use RDF Schema (RDFS) [13] and Web Ontology Language (OWL) [14] to interpret the data. Data in RDF Schema not only has literals but also a semantic meaning attached to it. This meaning has different informal hierarchies and formal taxonomies in its knowledge domain (space-domain). This leads to the introduction of a consensually shared view of concepts called Ontologies. Ontology is a formal, explicit specification of a shared conceptualisation [10]. The specifications use relations, functions, constraints, and axioms to conceptualise the abstract model. 'Formal', in the ontology definition, refers to the fact that the expressions must be machine readable; hence, natural language is excluded [6]. RDF was designed for situations where Web data need to be processed and exchanged by applications, rather than being displayed for people. The ability to exchange data between different applications means that the data may be made available to applications other than those for which they were originally intended [11].

Semantic data models and frameworks help organise the knowledge about specific domain and share amongst systems. The models help to see the "semantic" from different angles and refine the meaning by working on the ambiguity while reusing its commonality. When two (or more!) viewpoints come together in a web of knowledge, there will typically be overlap, disagreement, and confusion before synergy, cooperation and collaboration [1].

With Relational data on one side and RDF repository data on the other, the mapping algorithms have involved domain specific heuristic formulation to satisfy the accurate implementation of knowledge transfer. Different reasoning logic and rules are part of the mapping algorithms.

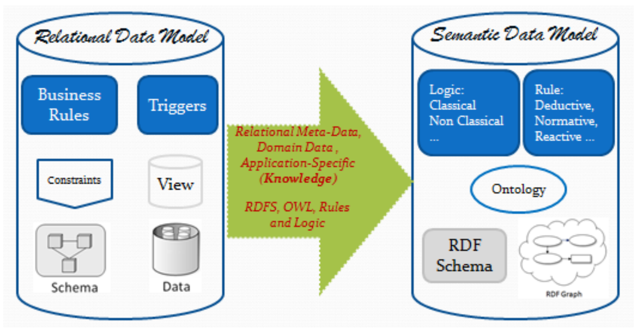

Relational Data Model
Business Rules
Triggers
Constraints
View
Schema
Data
Relational Meta-Data, Domain Data , Application-Specific (Knowledge)
RDFS, OWL, Rules and Logic
Semantic Data Model
Logic: Classical Non Classical ...
Rule: Deductive, Normative, Reactive ...
Ontology
RDF Schema
RDF Graph

Figure 1. Relational to Semantic Mapping Model

The mapping implementation involves RDF, Web Ontology Languages (RDFS/OWL), RDF query language (SPARQL) and semantic rule languages on Oracle database systems. The research project focuses on how to take advantage of the already existing vast relational data to build a knowledge base in using Semantic Web Framework. In the current Semantic architecture RDF is a favoured data serialization format to represent and manage knowledge. The ultimate goal of the research is a formal heuristics-based methodology for mapping relational database to semantic knowledge base. Section II explores existing approaches on mapping relational databases to RDF and points out what is being consistently missed during mapping. Section III analyses the different levels of 'knowledge' with respect to mapping. Sections IV and V outline the mapping algorithms and their implementation respectively and finally Section VI summarises with conclusion and future work.

II. EXISTING APPROACHES

The World Wide Web Consortium (W3C) RDB2RDF Working Group (WG) has conducted a survey of current approaches for mapping of relational databases to RDF [8]. The report summarises the different mapping implementation, query implementation, application domain, mapping creation, mapping representation and accessibility.

The mapping implementation is either using a static Extract Transform Load (ETL) or a dynamic on demand query-driven implementation. The static data warehousing approach has its own drawback in reflecting the current data. The queries are run in periodic intervals without compromising the current performance using mapping rules. It also gives an opportunity to analyse the data with respect to validation rules. The dynamic approach, on the other hand, costs a lot of performance time even though the outcome is a current reflection of what is in the relational database.

The query implementation follows two paths. The SPARQL-> RDF or the SPARQL-> SQL-> RDB path. SPARQL treats each RDF graph as a RDB table with three columns, ?subject, ?predicate and ?object. Each row corresponds to one tuple and the query result of SPARQL constitutes a table of RDF nodes [5].

Automatic mapping of RDB table to RDF class node and RDB column to RDF predicate leave behind most of the semantics of the data. Tools like Viruoso RDF View [3] expanded the above notion to map RDB unique identifier (primary key) to RDF object and column values as RDF subject. Even though these automatic mapping tools could be used as a starting point, there is still a lot to do to analyse, refine and process the Semantic data.

The survey also suggested the use of pre-existing public ontology resources such as the National Centre for Biomedical Ontologies [15] or an automatic domain-specific mapping tool such as D2RQ [2] that also allows custom user mapping rules. This approach also helps to reduce the amount of redundant knowledge. In one of the projects [4] based on the Royal Commission on the Ancient and Historical Monuments of Scotland (RCAHMS), 1.5 million entities of the database are converted into 21 million RDF triples. Using the domain semantics-driven generation the size of the RDF dataset is reduced by 2.8 million.

A further "feature-based comparison" between the major mapping languages (Direct Mapping, eD2R, R_2O, Relational.OWL, Virtuso, D2RQ, Triplify, R2RML, R3M) based on RDB2RDF WG report [8] also shows the different features of existing mapping languages [6]. The paper compares the mapping languages using four categories: *direct mapping, read-only general-purpose mapping, read-write general-purpose mapping,* and *special-purpose*.

What is being consistently missing from the existing mapping languages is the lack of "*knowledge*" consideration, which is not always explicitly represented and the use of "*rules*" and "*logic*". This "knowledge" can be *derived* from the explicitly represented relational model. It can also be *checked* using "application-specific predicates" and/or *executed* using "application-specific procedures/functions". The deficiencies of the existing mapping languages can be categorised into the following major levels of "knowledge".

1. Lack of using all the available "*Relational Database Area Knowledge"* and their variant meta-data combinations.

2. Lack of using "*Domain Data Knowledge*" like data-patterns (disjointness, symmetry, transitive chain, etc.)

3. Lack of using "*Application Specific Knowledge*" like "application-specific predicates" and "application-specific procedures/functions".

4. Lack of using "rules" and "logic" to elicit different "application-specific predicates" which is the recommendation of the W3C RDB2RDF WG [11].

III. RELATIONAL DATABASE KNOWLEDGE LEVELS

The mapping algorithms for converting relational databases into Semantic Web repositories which we have developed account several different types of knowledge: related to the relational model itself (*relational model knowledge*), related to the data stored in the database (*domain data knowledge*), related to the use of data (*domain users knowledge*) and knowledge about the database application (*application knowledge*). The conversion of the relational database is performed in three subsequent stages: *pre-processing*, during which the semantic repository is created and structured, *in-processing*, which incrementally maps the relational data and *post-processing*, which modifies the generated semantic repository to account additional domain-specific knowledge. After mapping the relational database to semantic RDF repository, different semantic rules are also applied to analyse the domain.

The domain knowledge which is broadly divided as "*relational database area knowledge*", "*domain data knowledge*", "*domain users knowledge*" and "*application-specific knowledge*" is used as a resource to facilitate the mapping of relational database to semantic RDF.

- Application Specific Knowledge
- Domain Data Knowledge
- Domain Users Knowledge
- Relational Database Area Knowledge

The use of the existing meta-data and knowledge at different levels as listed above and the formulation of additional knowledge from the existing knowledge contributes to the efficient representation of relational databases in Semantic Web.

A. *Relational Database area Knowledge*

The relational database area knowledge is used to identify the tables and columns to be considered in the mapping as well as the database constraints and data type restrictions on table columns. The relational database consists of different relational objects that are grouped into relational schemas. The tables and columns contain the data that is going to be mapped. In relational database, constraints are used to keep the integrity of the data. The constraints need to be mapped together with the data to maintain the integrity after mapping.

The database uses data type restrictions to guarantee data integrity during storing, retrieving and processing operations. The standard SQL data types are considered during the database manipulation process. These SQL data types are also mapped using an equivalent semantic RDF data type.

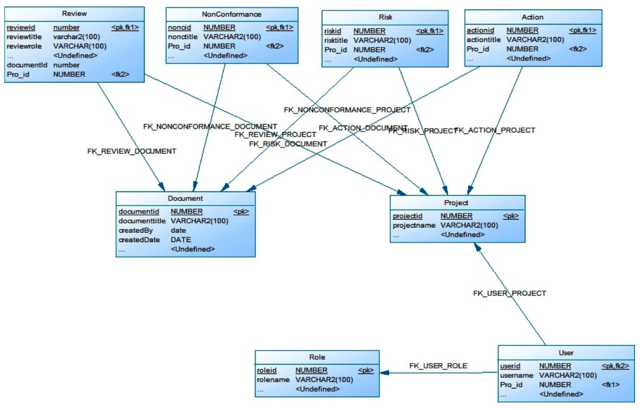


Figure 2. Relational Database Schema Overview

The relational database consists of database schemas (S) as a way of grouping relational objects. The database schema S(T1, T2, …, Tn), where n is the number of relational tables T is referred as the 'owner' of all the database objects under the schema. The relational data is stored in tables T(A1, A2, …, Am), where A is the column attribute and m is the number of column attributes in the table. Each table consists of different column attributes A1, A2, …, Am, where each column attribute has its own domain dom(A) and range ran(A).

The database constraints considered during mapping are primary key 'pk', foreign key 'fk', UNIQUE 'unq', NOT NULL 'nn', and CHECK 'ck' which are represented as pk(T), fk(T), unq(A), nn(A) and ck(A), respectively.

Each table T is a set of tuples t1, t2, …,tn, where n is the number of tuples in a table. Each tuple t is a set of values <v1, v2, …, vn>, where vi is the value corresponding to column attribute Ai for current tuple 1<=i<=n. Individual attribute values in a tuple are represented using the attribute and value pair as t(Ai, vi).

A relationship in relational databases is a situation that exists between two relational tables indicated by a foreign key constraint. The relationship between two tables is commonly referred as binary relationship. A group of binary relations may form a pattern that involve three (ternary), four (quaternary) or more tables that are commonly referred as N-ary relationships.

The tables involved in a relationship are classified as "strong" or "weak" tables depending on where the foreign key is placed. A "strong" table is indicated by a primary key (pk) database constraint using one or more column attributes while a "weak" table uses a foreign key to refer to the "strong" table.

Binary relationships are represented using a foreign key constraint that involve one or many cardinalities each side to form a one to one, one to many and many to many relationship.

- One-to-one (1:1)
- One-to-many (1:*)
- Many-to-many (*:*)

B. Domain Data Knowledge

Domain Data Knowledge describes how the domain data is used to represent knowledge in the mapped RDF Schema. Domain data is not represented explicitly but rather derived from the existing "*relational database area knowledge*".

Data patterns between relationships can be represented as domain knowledge using semantic web knowledge representation languages like OWL [14].

N-ary relationships that involve more than two tables create different patterns of relationships. These patterns are discovered using the primary and foreign key "*relational database area knowledge*" constraints on one or more columns. Patterns like "chain", "star", "triangular" help to identify more "*domain data Knowledge*" in addition to the "*relational database area knowledge*".

- Transitive Chain of Relations
- Disjointness
- Symmetries
- Star Relations, etc.

For instance, if there is a pattern where a database column is used both as a *primary key* 'pk' and a *foreign key* 'fk', then the referencing table is a "*subClass*" of the referenced table. This "*predicate*" uses primary key 'pk' and foreign key 'fk' "*relational database knowledg*e" to create a "subClass" relationship using a *symmetric*-type pattern on a column that is being used both as a primary key 'pk' and foreign key 'fk'.

```
Referencing Table T, Referenced Table T',
Column attribute A, primary key pk(T), foreign key fk(T)

Begin
  If( (A in (fk(T))) AND (A in (pk(T)))) then .

    <owl:Class rdf:ID="T" >
          <rdfs:subClassOf rdf:resource="#T'"/>
    </owl:Class>

  End if .
End .
```

Figure 3. Domain Data Knowledge "predicate"

C. Application Specific Knowledge

The 'Space Project Management' (SPP) domain-specific knowledge is used as a database schema source for evaluating the mapping of database application domain knowledge into semantic web. The domain-specific knowledge is used to 'prune' the semantic data at different stages of the *pre-processing* phase. The domain-specific knowledge is also a source for pattern discovery and interpretation at different stages of the *in-processing* and *post-processing* phase.

The applications used in SPP utilise different concepts and terminologies in the domain specific knowledge. The concepts that are used in the domain specific knowledge are summarised as *Documents, Risks, Non Conformances, Reviews, Actions,* and *Projects*. These concepts have unique as well as common domain specific knowledge. The unique domain specific knowledge is used to identify the concept while the common ones are used to associate and correlate the different domain specific concepts. The different concepts and sub-concepts establish domain specific SPP-Ontology.

Each "concept" has attributes that define and explain the concept. There are also sub-concepts that are related to the concept through relationships. Some attributes like "status type" are used to identify an application specific "status" within SPP-Ontology.

Both "concepts" and "Relationships" may have further sub-concepts and sub-relationships represented by a nested square bracket ([]).

- Attributes- [URI, Definition, Title, {Status}
- Status Types (sample)- [Register, Acknowledge, Assign Controller, Reduce, Accept, Resolve, Close]
- Sub-Concepts (sample)- [Domain, Scenario, Criticality [Likelihood, Severity], Rank, Trail]
- Relationship Types (sample)- [hasDomain, ofScenario, hasCriticality [hasLikelihood, hasSeverity], hasRank, hasTrail]

The concept URI attribute is a unique representation of the particular concept in SPP-Ontology. The "definition" and "title" attributes have the formal detailed definition and short descriptive title respectively.

The domain specific knowledge relies on attributes like "Relationship Type" to determine the semantic relationship between a "concept" and a "sub-concept" within SPP-Ontology.

For instance, if the application uses a common "Document Repository", an application-specific "predicate" can be used to check and relate all concepts to point to the document repository. This "*predicate*" is executed whenever the *primary key* 'pk' of "*Document*" concept is used as a *foreign key* 'fk' in the rest of the concepts.

IV. RDB-RDF MAPPING ALGORITHMS

Currently, the first phase of creating semantic RDF Schema ontology is finalised using the mapping procedures below. The procedures use the three layers of the use case knowledge- *"relational database area knowledge"*, *"domain data knowledge"* and *"application specific knowledge"*. After the data mapping, different levels of RDF inferencing will be applied to further explore the knowledge base.

1) mapDatabase

The database mapping procedure uses incremental iterative approach that loops through all tables T and their column attributes A within the schema S. The database mapping starts by mapping the tables- mapTable().

The tables are mapped into classes C. Corresponding to each class an RDF Repository C_RDF object is also created. The structure of the RDF repository is based on a TRIPLE format (subject, predicate, object).

After mapping the tables, mapColumn() algorithm maps the columns in each table into property column "hasP" of the existing class table C. mapColumn() procedure uses mapDatatype() procedure to return RDF equivalent data types for each relational attribute.

The different relational database constraints are also mapped using mapConstraint() procedure. The procedure maps constraints like primary key 'pk', foreign key 'fk', UNIQUE 'unq', NOT NULL 'nn', and CHECK 'ck'.

Finally, the algorithm maps the different relationships among tables and columns. MapRelationship() algorithm uses the foreign key constraint between tables to find out the different types of relationship with respect to degree of relationship, transitive chain of relation, disjointness, etc.

```
Procedure mapDatabase (S)
    Input: Schema S
Begin

    mapTable(S) .
    mapColumn(S) .
    mapConstraint(S) .
    mapRelationship(S) .

End .
```

Figure 4. mapDatabase() Algorithm

2) mapTable

The Semantic Web equivalent of the relational algebra relations is a Class. During mapping the same name for the table T is used for the mapped class C. The class name is used to map subsequent relational columns into semantic class properties.

The classes in the Semantic Web repository can be considered repositories of data to hold the relational data after the mapping. Each Class table C represents the relational table in semantic web. To represent the class data in RDF triples (subject, predicate, object), a separate RDF repository C_RDF is created. In C_RDF the class ID (i.e., primary key equivalent of the source table) is used as a 'subject' while the rest of the properties are used as 'predicates'. Each property "value" corresponding to the class ID is the 'object' of the RDF triple.

In addition to the class table C and RDF repository C_RDF, an OWL class is created using the class table C as an ID.

```
Procedure mapTable (S)
    Input: Schema S
    Output: Class C, RDF Repository C_RDF, OWL Class
Begin
    For each table Ti in S loop .

        Create Class Table Ci .

        Create RDF Repository Ci_RDF
        using Class Table Ci and  a TRIPLE type attribute .

        <owl:Class rdf:ID="Ci" />

    End loop .
End .
```

Figure 5. mapTable() Algorithm

3) mapColumn

The column attributes in the relational database are mapped as properties in semantic classes. A property in a class can describe an entity class or a relationship class. Each property has a set of allowable domain values that could be shared with one or more properties.

The columns are broadly divided as "key columns" that are used to identify an occurrence of a relation and "simple columns" that only describe a relation. During mapping columns into properties, the following column types are used as criteria to choose the appropriate representation in the semantic web.

Column types

- Candidate Key (CK): minimal set of attributes that uniquely identifies each occurrence of an entity type.
- Primary Key (PK): candidate key selected to uniquely identify each occurrence of an entity type.
- Foreign Key (FK): referencing a primary key (PK) in another relation
- Simple Column (SC): a non-candidate key that describes an entity type.

The column attributes of a table is denoted as T(A1, A2, …, An)
T(A1, A2, …, An) = {PK, {CK1, CK2,…, CKx}, {FK1, FK2, …, FKy}, {SC1, SC2, …, SCz}}
Where x, y, z is a whole number.
When the cardinality of x is greater than 1, candidate keys (CK) are treated as *composite* keys.

The column attributes in the relational database are mapped to corresponding class properties. The constraint type associated with the attribute determines the cardinality. Each column attribute A is mapped to property P prefixed by the word "has" as "hasP".

```
Procedure mapColumn (S)
    Input: Schema S, Table T, Column attribute A

    Output: Property P, OWL:DatatypeProperty
Begin
```

```
  For each table Ti in S loop .
    For each Column Aj in Ti loop .
                    get mapped Class Table Ci of Ti.
                              set Aj as Property Column hasAj .
                              get_&xsd;type_equivalent (Aj) .
                    <owl:DatatypeProperty rdf:ID="hasAj">
                    <rdfs:domain rdf:resource="#Ci" />
                    <rdfs:range rdf:resource
                      ="&xsd;type_equivalent" />
                    </owl:DatatypeProperty>
          End loop .
    End loop .
End .
```

Figure 6. mapColumn() Algorithm

4) mapConstraint

mapConstraint() algorithm maps relational database constraints into their equivalent semantic web representation. The algorithm reads both table level as well as column level constraints. For primary key pk(T) constraints, it creates "InverseFunctionalProperty" and "maxCardinality" OWL properties. For foreign key fk(T) constraints, it creates "ObjectProperty" OWL property. If a foreign key column is also part of the primary key pk(T) constraints, then the referencing table (T) is set as a "subClass" of the referenced table (T').

For UNIQUE 'unq(A)', NOT NULL 'nn(A)', and CHECK 'ck(A)' database constraints, the algorithm creates equivalent and "InverseFunctionalProperty", "minCardinality", and "hasValue" OWL property restrictions respectively.

If the column attribute is a primary key pk(T) of the table, the maximum cardinality of the property car(P) is set to one. If the column attribute has a unique constraint unq(A), the maximum cardinality of the property car(P) is set to one. On the other hand if the property has a NOT NULL constraint nn(A), the minimum cardinality of the property car(hasP) is set to one.

mapConstraint() procedure maps column attribute(s) A using the table T and its constraints (primary key, foreign key, UNIQUE, NOT NULL, CHECK) to a semantic property P and OWL cardinality properties.

```
Procedure mapConstraint (S)
    Input: Schema S, Table T, Referenced Table T', Column
    attribute A, primary key pk(T), foreign key fk(T),
    UNIQUE unq(A), NOT NULL nn(A), and CHECK
    ck(A)
    Output: RDFS subClassOf, Property P, OWL cardinality
    properties
Begin
For each table Ti in S loop .
 For Column Aj in Ti loop .
  get mapped Class Table Ci of Ti.

  If (Aj in (pk(Ti))) then .

   <owl:InverseFunctionalProperty rdf:resource="# hasAj "/>

     /* set maximum car(hasAj) to 1 . */
     <rdfs:subClassOf>
       <owl:Restriction>
           <owl:maxCardinality
             rdf:datatype="&xsd:nonNegativeInteger">1
           </owl:maxCardinality>
       </owl:Restriction>
     </rdfs:subClassOf>

  Else
  if (Aj in (fk(Ti))) then .
     If (Aj in (pk(T'i))) then .
       <rdfs:subClassOf rdf:resource="#C'"/>
     End if .
           <owl: ObjectProperty rdf:ID="hasA">
           <rdfs:domain rdf:resource="#C" />
           <rdfs:range rdf:resource="#C'" />
           </owl: ObjectProperty >
  Else
  if (unq(Aj)) then .

   <owl:InverseFunctionalProperty rdf:resource="# hasAj "/>

  Else
  if (nn(Aj) and (!pk(Aj)) then .
           /*set minimum car(hasAj) to 1 .*/
           <owl:Restriction>
                 <owl:minCardinality
                 rdf:datatype="&xsd:nonNegativeInteger">1
                 </owl:minCardinality>
           </owl:Restriction>

  Else
  if (ck(Aj)) then .
     <rdfs:subClassOf>
     <owl:Restriction>
           <owl:onProperty rdf:resource="#hasAj" />
           <owl:hasValue rdf:datatype="&xsd;string" > v(Aj)
           </owl:hasValue>
     </owl:Restriction>
     </rdfs:subClassOf>
  End if .
 End loop .
End loop .
End .
```

Figure 7. mapConstraint() Algorithm

5) mapRelationship

A relationship between tables is identified by a foreign key. MapRelationship() calls a series of sub-procedures to identify the type of relationship between the two tables and discover any patterns with the rest of the tables in the schema.

```
Procedure mapRelationship (S)
    Input: Table T, Column attribute A, foreign key fk(T)
    Output: Class C, Property P
Begin
    For each table Ti in S loop .

        For each column Aj in Ti loop .
            If (Aj = fk(Ti)) then
                CheckRelationship(Ti, T'i)
                CheckTransitiveChain(Ti, T'i) .
                CheckDisjointness(Ti, T'i)

            where T'i is referenced table by Ti
            End if .
        End loop .
    End loop .
End .
```

Figure 8. mapRelationship() Algorithm

6) Check Relationship

CheckRelationship algorithm uses the referencing table T and the referenced table T' to determine whether to create a separate class to represent the relationship or only add a property column to the existing class.

If the foreign key fk(T) is referring to a primary key pk(T') and the foreign key has a NOT NULL constraint, then a new relationship class is created using the two tables (T, T'). An additional "subClass" parameter is also passed to create a "subClass" axiom between the "Class" equivalents of the tables T and T'.

If the above criterion is not satisfied, the foreign key fk(T) would have been added as a property to the referring table equivalent class C using the mapColumn() procedure above.

```
Procedure CheckRelationship(T, T')
    Input: Table T, primary key pk(T), foreign key
    fk(T),NOT NULL nn(T)

Begin
    If (fk(T) = pk(T') and (fk(T) = nn(fk(T))) then .

        CreateRelationship(T,T', subClass) .

    End if .
End .
```

Figure 9. CheckRelationship() Algorithm

7) Check Transitive Chain

Transitive Chain of relations is tested using the foreign key/primary key column attributes between three relational tables.

For any relational tables T1, T2, T3 in Schema S, if there is a foreign key relationship between T1 and T2 and if there is also a foreign key relationship between T2 and T3, then there is a transitive chain between T1 and T3.

The algorithm uses the referenced table's (T') columns to find further foreign key relationship to the rest of the tables. If another relationship other than the one between T and T' is found, a new relationship class is created using the two tables (T, Ti). An additional "Transitive" parameter is also used to create a "transitive" axiom between the "Class" equivalents of the starting table T and the new third table Ti.

```
Procedure CheckTransitiveChain(T, T')
    Input: Table T, Column attribute A, primary key pk(T),
    foreign key fk(T)
Begin

    For each column Ai in T' loop .
        If (Ai in fk(T')) then .
        For each table Ti in S loop .
            If ((Ai in pk(Ti)) and (Ti != T)) then .

            createRelationship(T, Ti, Transitive) .

            End if .
        End loop .
        End if .
    End loop .
End .
```

Figure 10. CheckTransitiveChain() Algorithm

8) Check Disjointness

Disjointness is a relationship between two "SubType" tables that share a common "SuperType" but has no relationship between each other.

The foreign key/primary key column attributes between the tables is used to determine the disjunction between tables.

For any relational tables T1, T2, T3 in Schema S, if there is a foreign key relationship between T1 and T2 and there is also a foreign key relationship between T2 and T3 but there is no relationship between T1 and T3, then there is a disjointness between T1 and T3.

The algorithm uses the referenced table's (T') columns to find further foreign key relationship to other tables. If another relationship other than the one between T and T' is found and there is no foreign key relationship between the new table Ti and the starting table T, then a "disjointWith" axiom is created between the starting table T and the new table Ti.

Note that a group of disjoint relationships create a "Star" relation with the "SuperClass" in the middle and the disjoint classes as a branch.

```
Procedure CheckDisjointness(T, T')
    Input: Table T, Column attribute A, primary key pk(T),
    foreign key fk(T)
    Output: RDFS subClassOf, OWL Class, disjointWith
Begin

    For each column Ai in T' loop .
        If (Ai in fk(T')) then .
                For each table Ti in S loop .
                If ((Ai in pk(Ti)) and (Ti != T)) then .
                        if ((ALL) fk(Ti) NOT in
                            (ALL) pk (T)) then .

                <owl:Class rdf:ID="Ti">

                <rdfs:subClassOf rdf:resource="#T"/>

                <owl:disjointWith rdf:resource=" #T "/>
                </owl:Class>

                End if .
                End loop .
        End if .
    End loop .

End .
```

Figure 11. CheckDisjointness() Algorithm

9) *Create Relationship*

CreateRelationship algorithm is used to create a new relationship class table to represent the relationship between the referencing table T and the referenced table T'. If there is a foreign key in table T that references to a primary key in table T', a new class table is created using a class symbol C and the name of the referencing and referenced tables respectively separated by an underscore as "C_T_T' ".

The primary keys of both tables are added as property columns to the new class by adding "has" as a prefix as "hasP" and "hasP' ".

In addition to the semantic class, a repository is also created to represent the class table data in RDF triples (subject, predicate, object). The RDF repository reads the class table data and presents it in RDF triples. It is named using the class table names suffixed by "RDF" as "C_T_T'_RDF".

```
Procedure CreateRelationship(T, T', TYPE)
    Output: RDFS subClassOf, OWL Class, ObjectProperty
Begin
    If (fk(T) = pk(T')) then .

        create Class C _T_T'.
        set pk(T) as Property hasP of Class C_T_T' .
        set pk(T') as Property hasP' of Class C_T_T' .

        create RDF Repository C_T_T'_RDF
                using Class Table C_T_T' and a TRIPLE
                    type attribute .

                get mapped Class Table C of T .
                get mapped Class Table C' of T'.

        If (TYPE = 'subClass') then
                <owl:Class rdf:ID="C">
                        <rdfs:subClassOf
                            rdf:resource="#C'" />
                </owl:Class>

        Else
        if (TYPE = 'Transitive') then

                <owl:ObjectProperty rdf:ID="pk(T)">
                <rdf:type rdf:resource
                            ="owl;TransitiveProperty"/>
                <rdfs:domain rdf:resource="#C" />
                <rdfs:range rdf:resource="#C'" />
                </owl:ObjectProperty>

        End if .

End .
```

Figure 12. CreateRelationship() Algorithm

V. IMPLEMENTATION

The implementation of the algorithms is based on the Space Project Management scenario. An interactive Java application is used to execute the mapping procedures and the resulting Semantic Web repository loaded in Protégé (a free open-source Java tool providing an extensible architecture for the creation of customized knowledge-based applications) OWL editor is shown on the figure below.

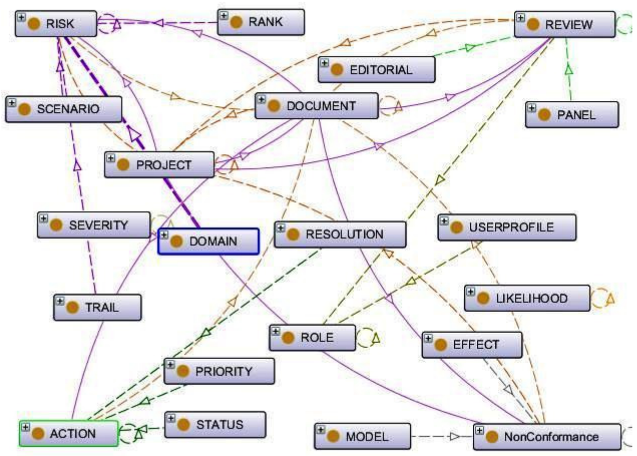
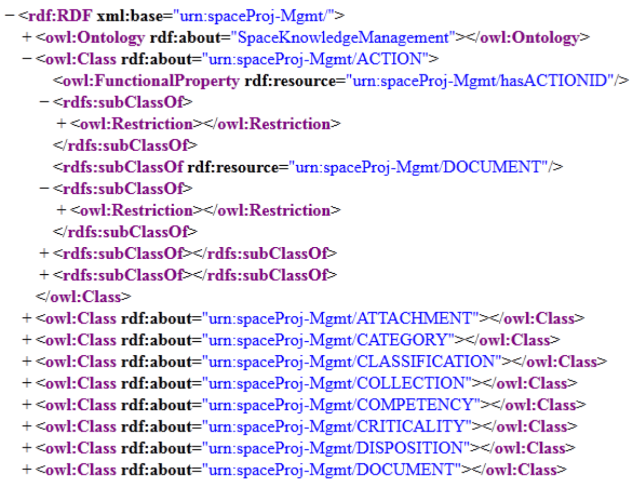


Figure 13. RDF Schema view (Protége)

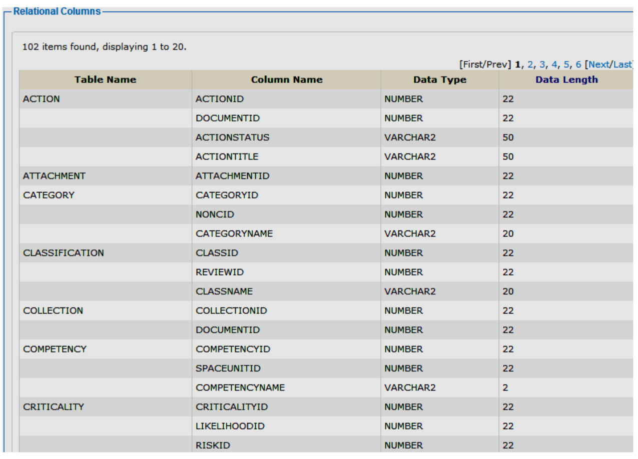
The Ontology diagram modeled above is also serialisable using OWL file format as in Figure 14 below.



```
− <rdf:RDF xml:base="urn:spaceProj-Mgmt/">
  + <owl:Ontology rdf:about="SpaceKnowledgeManagement"></owl:Ontology>
  − <owl:Class rdf:about="urn:spaceProj-Mgmt/ACTION">
      <owl:FunctionalProperty rdf:resource="urn:spaceProj-Mgmt/hasACTIONID"/>
    − <rdfs:subClassOf>
       + <owl:Restriction></owl:Restriction>
      </rdfs:subClassOf>
      <rdfs:subClassOf rdf:resource="urn:spaceProj-Mgmt/DOCUMENT"/>
    − <rdfs:subClassOf>
       + <owl:Restriction></owl:Restriction>
      </rdfs:subClassOf>
    + <rdfs:subClassOf></rdfs:subClassOf>
    + <rdfs:subClassOf></rdfs:subClassOf>
    </owl:Class>
  + <owl:Class rdf:about="urn:spaceProj-Mgmt/ATTACHMENT"></owl:Class>
  + <owl:Class rdf:about="urn:spaceProj-Mgmt/CATEGORY"></owl:Class>
  + <owl:Class rdf:about="urn:spaceProj-Mgmt/CLASSIFICATION"></owl:Class>
  + <owl:Class rdf:about="urn:spaceProj-Mgmt/COLLECTION"></owl:Class>
  + <owl:Class rdf:about="urn:spaceProj-Mgmt/COMPETENCY"></owl:Class>
  + <owl:Class rdf:about="urn:spaceProj-Mgmt/CRITICALITY"></owl:Class>
  + <owl:Class rdf:about="urn:spaceProj-Mgmt/DISPOSITION"></owl:Class>
  + <owl:Class rdf:about="urn:spaceProj-Mgmt/DOCUMENT"></owl:Class>
```

Figure 14- RDF Schema Overview (OWL file excerpt)

The mapping test between the Relational Database Schema (Figure 2) and RDF Schema (Figure 13 & Figure 14) is evaluated using meta-data search comparisons as shown by the sample screenshots in Figure 15 & Figure 16 below.



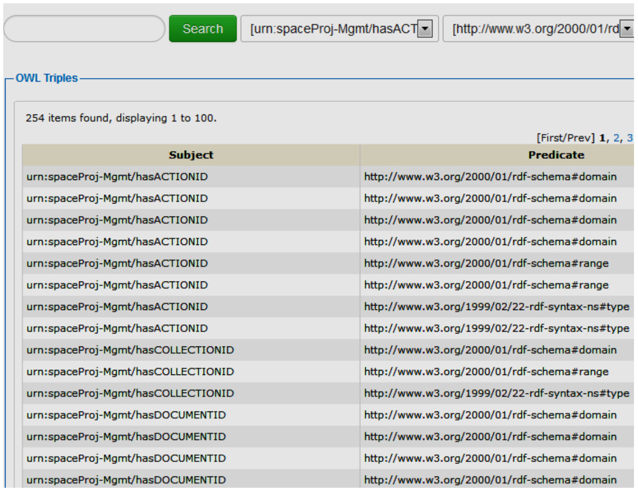
Relational Columns

102 items found, displaying 1 to 20.

[First/Prev] 1, 2, 3, 4, 5, 6 [Next/Last

Table Name	Column Name	Data Type	Data Length
ACTION	ACTIONID	NUMBER	22
	DOCUMENTID	NUMBER	22
	ACTIONSTATUS	VARCHAR2	50
	ACTIONTITLE	VARCHAR2	50
ATTACHMENT	ATTACHMENTID	NUMBER	22
CATEGORY	CATEGORYID	NUMBER	22
	NONCID	NUMBER	22
	CATEGORYNAME	VARCHAR2	20
CLASSIFICATION	CLASSID	NUMBER	22
	REVIEWID	NUMBER	22
	CLASSNAME	VARCHAR2	20
COLLECTION	COLLECTIONID	NUMBER	22
	DOCUMENTID	NUMBER	22
COMPETENCY	COMPETENCYID	NUMBER	22
	SPACEUNITID	NUMBER	22
	COMPETENCYNAME	VARCHAR2	2
CRITICALITY	CRITICALITYID	NUMBER	22
	LIKELIHOODID	NUMBER	22
	RISKID	NUMBER	22

Figure 15. Relational Database Schema search result



OWL Triples

254 items found, displaying 1 to 100.

[First/Prev] 1, 2, 3

Subject	Predicate
urn:spaceProj-Mgmt/hasACTIONID	http://www.w3.org/2000/01/rdf-schema#domain
urn:spaceProj-Mgmt/hasACTIONID	http://www.w3.org/2000/01/rdf-schema#domain
urn:spaceProj-Mgmt/hasACTIONID	http://www.w3.org/2000/01/rdf-schema#domain
urn:spaceProj-Mgmt/hasACTIONID	http://www.w3.org/2000/01/rdf-schema#domain
urn:spaceProj-Mgmt/hasACTIONID	http://www.w3.org/2000/01/rdf-schema#range
urn:spaceProj-Mgmt/hasACTIONID	http://www.w3.org/2000/01/rdf-schema#range
urn:spaceProj-Mgmt/hasACTIONID	http://www.w3.org/1999/02/22-rdf-syntax-ns#type
urn:spaceProj-Mgmt/hasACTIONID	http://www.w3.org/1999/02/22-rdf-syntax-ns#type
urn:spaceProj-Mgmt/hasCOLLECTIONID	http://www.w3.org/2000/01/rdf-schema#domain
urn:spaceProj-Mgmt/hasCOLLECTIONID	http://www.w3.org/2000/01/rdf-schema#range
urn:spaceProj-Mgmt/hasCOLLECTIONID	http://www.w3.org/1999/02/22-rdf-syntax-ns#type
urn:spaceProj-Mgmt/hasDOCUMENTID	http://www.w3.org/2000/01/rdf-schema#domain
urn:spaceProj-Mgmt/hasDOCUMENTID	http://www.w3.org/2000/01/rdf-schema#domain
urn:spaceProj-Mgmt/hasDOCUMENTID	http://www.w3.org/2000/01/rdf-schema#domain
urn:spaceProj-Mgmt/hasDOCUMENTID	http://www.w3.org/2000/01/rdf-schema#domain

Figure 16. RDF Triples search result

We have compared our attempt to some of the existing approaches (Direct Mapping, eD2R, R2O, Relational.OWL, Virtuso, D2RQ, Triplify, R2RML, R3M). We used the comparison criteria specified in the seminal paper [6]. The results of this comparison can be summarized as shown in Table I and Table II.

TABLE I. RDB-TO-RDF MAPPING LANGUAGES COMPARISON LEGEND [6]

Legend	Description
F1	Logical Table to Class
F2	M:N Relationships
F3	Project Attributes
F4	Select Conditions
F5	User-dened Instance URIs
F6	Literal to URI
F7	Vocabulary Reuse
F8	Transformation Functions
F9	Datatypes
F10	Named Graphs
F11	Blank Nodes
F12	Integrity Constraints
F13	Static Metadata
F14	One Table to n Classes
F15	Write Support
F16	**Data Patterns**
F17	**Data Control Languages (DCL)**

TABLE II. SUMMARY TABLE OF RDB-TO-RDF MAPPING LANGAGE COMPARISON [6] WITH RD2SW

	F1	F2	F3	F4	F5	F6	F7	F8	F9
Direct Mapping	(✓)	×	×	×	×	×	×	×	×
eD2R	✓	✓	✓	✓	✓	✓	✓	✓	✓
R_2O	✓	×	✓	✓	✓	✓	✓	✓	✓
Relational.OWL	(✓)	×	✓	×	×	×	×	×	✓
Virtuoso	✓	✓	✓	✓	✓	✓	✓	✓	✓

D2RQ	✓	✓	✓	✓	✓	✓	✓	✓	✓
Triplify	✓	✓	✓	✓	✓	✓	✓	✓	✓
R2RML	✓	✓	✓	✓	✓	✓	✓	✓	✓
R3M	(✓)	✓	✓	(✓)	✓	✓	✓	✓	✓
RD2SW	✓	✓	✓	✓	✓	✓	✓	✓	✓

	F10	F11	F12	F13	F14	F15	F16	F17
Direct Mapping	✗	✗	✗	✗	✗	✓		
eD2R	✗	✓	(✓)	✗	✓	✗		
R_2O	✗	✓	(✓)	✗	(✓)	✗		
Relational.OWL	✗	✓	(✓)	✗	✗	✓		
Virtuoso	✓	✓	(✓)	✗	✓	✗		
D2RQ	✗	✓	(✓)	✓	✓	✗		
Triplify	✗	✗	(✓)	✗	✓	✗		
R2RML	✓	✓	(✓)	✓	✓	✗		
R3M	✗	✗	✓	✗	✓	✓		
RD2SW	✗	✓	✓	✗	✓	✗	✓	✓

VI. CONCLUSION AND FUTURE WORK

In the current implementation, the mapping procedure reads the database directly, starting with the data dictionary and then the data. The developed algorithms use a configuration file to choose a relational database driver and access the database dictionary to map into an RDF format. It involves formulation of heuristics that formally define the mapping. The domain specific heuristics have helped harvest the Ontology of the Space Project Management Database. This domain specific heuristics was implemented using an incremental algorithm to extend the domain ontology repository. The formulated heuristics and domain ontology repository have been implemented and tested on a prototype Space Project Management semantic tool as a proof-of-concept to the research. This heuristic-based methodology can be applied and measured on other relational data with different domain ontology.

Currently, we used set of heuristics for accounting the different types of *relational model knowledge* (constraints, data types), *domain specific knowledge* (simple data patterns like transitive chain and disjointness) and *application specific knowledge* (predicates). In the future we plan to extend the algorithm so that it also accounts other types of *domain specific knowledge* like complex data-patterns, *user domain knowledge* (individual and group users, access rights and profiles) and *application domain knowledge* (i.e., triggers and transactions). We also plan to implement a parser for SQL DDL, used to create the database. It will be still necessary to connect to the database in order to elicitate and convert the data stored in it, but this will eliminate the need for using the data dictionary and thus, it will reduce the database dependency.

The process of mapping Relational Databases to Semantic Web (RD2SW) using domain specific knowledge involves most of the current - Semantic Web Layer components- RDF, RDF schema, query languages, rules, logic, etc. Following the Semantic Web standards set by World Wide Web Consortium will ultimately help us represent 'web resources' in a standardized, unambiguous, interoperable and above all Linked-Data format as the next efficient phase of representing knowledge in the 21st century.

ACKNOWLEDGMENT

The authors acknowledge the sponsorship of this PhD research work by the London Metropolitan University Vice Chancellor's fund. Implementation and evaluation is conducted in collaboration with Sapienza Consulting [16] a leading software provider for space mission and project support.

REFERENCES

[1] D. Allemang and J. Hendler, Semantic Web for the Working Ontologist: Effective Modeling in RDFS and OWL. Morgan Kaufmann Publishers, CA, 2008

[2] C. Bizer and R. Cyganiak., "D2RQ Lessons Learned" Position paper for the W3C Workshop on RDF Access to Relational Databases, Cambridge, MA, USA, October 2007, pp. 25-26.

[3] C. Blakeley, "RDF Views of SQL Data (Declarative SQL Schema to RDF Mapping)" OpenLink Software, 2007.

[4] K. Byrne, "Having Triplets Holding Cultural Data as RDF" Proceedings of the ECDL 2008 Workshop on Information Access to Cultural Heritage, Aarhus, Denmark, September 2008.

[5] R. Cyganiak, "A relational algebra for SPARQL" Digital Media Systems Laboratory HP Laboratories Bristol. HPL-2005-170, 2005.

[6] M. Hert, G. Reif, and H. Gall, "A comparison of RDB-to-RDF mapping languages" In Proceedings of the 7th International Conference on Semantic Systems, Graz, Austria, ACM, 2011, pp. 25-32.

[7] S. Staab and R. Studer, Handbook on Ontologies. Springer Berlin: 2009.

[8] S.S. Sahoo, W. Halb, S. Hellmann, K. Idehen, T. Thibodeau, S. Auer, J. Sequeda, and A. Ezzat, "A Survey of Current Approaches for Mapping of Relational Databases to RDF" W3C RDB2RDF XG Incubator Report: W3C, 2009.

[9] A. Sheth, C. Ramakrishnan,and C. Thomas, "Semantics for the Semantic Web: The Implicit, the Formal and the Powerful" Int'l Journal on Semantic Web & Information Systems, 2005, pp. 1-18.

[10] R. Studer,V. R. Benjamins, and D. Fensel, "Data and Knowledge Engineering" Knowledge engineering: Principles and methods, 1998, pp.161-197.

[11] E. Marx, P. Salas, K. Breitman, and J. Viterbo, "RDB2RDF: A relational to RDF plug-in for Eclipse" Published online in Wiley Online Library (wileyonlinelibrary.com), 2012 [retrieved: December, 2012].

[12] F. Manola, E. Miller, and B. McBride, "RDF Primer" W3C Recommendation, Feb. 2004.

[13] R.V. Guha and B. McBride, "RDF Vocabulary Description Language 1.0: RDF Schema" W3C Recommendation, Feb. 2004.

[14] D.L. McGuinness and Frank Van Harmelen, "OWL web ontology language overview." W3C recommendation, Feb 2004.

[15] NCBO BioPortal, The National Center for Biomedical Ontology, [retrieved: December, 2012] from http://bioportal.bioontology.org

[16] Sapienza Consulting, About Sapienza, [retrieved: December, 2012] from http://www.sapienzaconsulting.com