R2D: Extracting Relational Structure from RDF Stores

Sunitha Ramanujam¹, Anubha Gupta¹, Latifur Khan¹, Steven Seida², Bhavani Thuraisingham¹
¹The University of Texas at Dallas
²Raytheon Corporation
{sxr063200, axg089100, lkan, bxt043000}@utdallas.edu

Abstract – The enthusiastic acceptance of Resource Description Framework (RDF) as a data model has given birth to a new data storage paradigm, namely, the RDF Graph model. The pool of modeling and visualization tools available for RDF stores is limited due to the technology being in its fledgling stage. The work presented in this paper, called R2D (RDF-to-Database) is an effort to make available, to RDF data stores, the abundance of relational tools that are currently in the market. This is done in the form of a JDBC wrapper around RDF Stores that presents a relational view of the stores and their data to the modeling and visualization tools. This paper presents key R2D functionalities and mapping constructs, procedures for every stage of R2D deployment, and sample results in the form of screenshots and performance graphs.

Keywords: Semantic Web, Resource Description Framework, Relational Databases, Data Interoperability

I. INTRODUCTION

In recent years, the explosion of the Internet has resulted in the emergence of an evolutionary stage of the World Wide Web, namely, the Semantic Web. To realize the Semantic Web vision various standards, such as Resource Description Framework [1], are being developed to enable users to access information more efficiently and accurately. The simplicity and flexibility offered by RDF data models have resulted in an increase in the number of data stores that use the RDF Graph model.

Such a plethora of RDF information stores have, consequently, given rise to the need for tools to manage and visualize this data. However, most of the currently available data modeling, visualization, and management tools are still based on the more mature models such as relational and tabular models [2]. In order to continue to leverage the advantages offered by relational tools without losing out on the benefits offered by newer web technologies, the gap between the two needs to be bridged.

One method to bridge this gap is to create an equivalent relational schema in an existing Relational Database Management System and copy the RDF triples data into the corresponding tables in the relational schema. This approach leads to space wastage due to duplication of the data in the RDF store. Further, synchronization of the data in the two stores is another issue to be considered, and some sort of resource and time intensive mechanism would have to be in place to ensure that the relational store is a true and current version of the RDF store.

We propose a solution to the bridging problem without the need to create an actual physical relational schema and duplicate data. The work presented in this paper, called R2D (RDF-to-Database), is a bridge that hopes to enable existing relational tools to work seamlessly with RDF Stores without having to make extensive modifications or waste valuable resources by replicating data unnecessarily. Our research provides a relational interface to data stored in the form of RDF triples and, to the best of our knowledge, has no comparable counterparts. Our contributions are:

- We propose a mapping scheme for the translation of RDF Graph structures to an equivalent relational schema
- The proposed mapping process includes the ability to map, through extensive examination of instance data, even “sloppy” RDF Graphs that either do not have any, or have incomplete structural/schema information included along with the data.
- Based on the RDF-to-RDBMS map file created, we propose a transformation process that presents a normalized, non-generic, domain-specific, virtual relational schema view of the given RDF store
- We propose a mechanism to transform any relational SQL queries issued against the virtual relational schema into its SPARQL equivalent, and return the triples data to the end-user in a relational/tabular format
- We provide all of the above in the form of a JDBC interface that can be plugged into existing visualization tools seamlessly.

Section II provides an overview of current research in the relational-to-rdf arena. Section III discusses R2D’s modus operandi and mapping constructs. The algorithms involved in the mapping process are described in Section IV followed by experimental results in Section V. The paper concludes with a discussion on the advantages of this research in Section VI.

II. RELATED WORK

While the overall concept of R2D is unique and has no comparable counterparts, several research efforts exist that
attempt to bring relational database concepts and semantic web concepts together, albeit from a perspective that is opposite to that considered in our work. Some of these efforts include D2RQ [3] and Virtuoso RDF Views [4] which are essentially mapping efforts between relational schema and OWL/RDFS concepts where a relational database schema is taken as input and an RDF interface of the same is presented as output. Tripify [5] is another effort at publishing linked data from relational databases and it achieves this by extending SQL and using the extended version as a mapping language. RDF123 [6], an open source translation tool, also uses a mapping concept, however its domain is spreadsheet data and it attempts to achieve spreadsheet-to-RDF translation by allowing users to define mappings between spreadsheet semantics and RDF graphs.

The Hybrid model [7] is the nearest match to the mapping methodology in our work, however, since the model generates a table for every property in the ontology, it results in unnecessary tables in the case of 1:N relationships between subject and object resources. R2D avoids this by adding a foreign key column to the appropriate table when processing 1:N relationships. The hybrid model also fails on RDF graphs which do not include schema information while R2D is able to glean structural information even in the absence of ontological constructs.

As can be seen from the above discussions, none of the existing research efforts address the issue of enabling relational applications to access RDF data without data replication. Thus, to the best of our knowledge, R2D is the first endeavor to address this issue.

III. R2D PRELIMINARIES

The architecture of the proposed system and the deployment sequence of the algorithms comprising R2D are illustrated in Figure 1. R2D’s functionality is made available as a JDBC Interface that can be plugged into any visualization tool that is based on a relational data model.

Figure 1. R2D Architecture & Deployment Sequence

Table 1 tabulates the notional mapping between OWL/RDFS Ontology terminologies and relational concepts that is adopted by R2D.

<table>
<thead>
<tr>
<th>OWL/RDFS</th>
<th>RELATIONAL CONCEPT</th>
</tr>
</thead>
</table>

Table 1: Notional Mapping between RDFS/OWL and R2D

At the heart of the relational transformation of RDF Graphs is the R2D mapping language – a declarative language that expresses the mappings between RDF Graph constructs and relational database schema constructs. In order to better explain the R2D mapping language constructs, examples from the sample scenario in Figure 2 are included where applicable.

Figure 2. Sample Scenario

The constructs of the current version of the mapping language are presented below.

**r2d:TableMap:** The r2d:TableMap construct refers to a table in a relational database. In most cases, each rdfs:class object will map to a distinct r2d:TableMap, and, in the absence of rdfs:class objects, the r2d:TableMaps are inferred from the instance data in the RDF Store. **Example:** The RDF Graph in Figure 2 results in the creation of a TableMap called “Student”.

The mapping constructs specific to an r2d:TableMap are as follows.  

**r2d:keyField:** The r2d:keyField construct specifies the primary key attribute for the r2d:TableMap to which the field is attached. The data value associated with the field specified by r2d:keyField is the object of the “rdfs:type” predicate belonging to the rdfs:class subject corresponding to its r2d:TableMap. **Example:** An r2d:keyField (primary key) called “Student_PK” is field attached to the “Student” TableMap and one of its values, corresponding to the sample scenario in Figure 2, is “URI/StudentA”.

**r2d:ColumnBridge:** r2d:ColumnBridges relate single-valued RDF Graph predicates/properties to relational database columns. Each rdfs:property object maps to a distinct column attached to the table specified in the rdfs:domain predicate. In the absence of
Example: The “Name” and “Member Of” predicates in Figure 2 become r2d:ColumnBridges belonging to the “Student” r2d:TableMap

r2d:MultiValuedColumnBridge(MVCB): Those RDF predicates that have multiple object values for the same subject are mapped using the MVCB construct. MVCBs typically correspond to RDF constructs such as RDF containers and collections and are used to indicate N:1 and N:M relationships between the virtual relational schema tables.

Example: The “Works On” predicate in Figure 2 is an example of an MVCB mapping.

r2d:SingleValuedBlankNode (SVBN): This construct helps relate blank nodes with distinct predicates to relational database columns. In the virtual relational schema, the blank node is ignored and the predicates of blank nodes are treated as having simple 1:1 relationships to the subject of the blank node.

Example: The object of the “Address” predicate in Figure 2 is an example of an SVBN that has the distinct predicates of “Street”, “City”, and “State”.

r2d:MultiValuedBlankNode (MVBN): This construct refers to blank nodes in the RDF Graph that contain repeating predicates. These blank nodes have multiple object values for the same subject and predicate concept associated with the blank node. An MVBN typically results in the generation of a separate r2d:TableMap with a foreign key relationship.

Example: The object of the “Phone” predicate in Figure 2 is an example of an MVBN that has multiple object (Cell) values for the subject (URI/StudentA) and predicate (Cell) concept associated with the MVBN.

The mapping constructs specific to column bridges and blank nodes are described below.

r2d:belongsToTableMap(BTTM): This construct connects a r2d:ColumnBridge or MVCB to an r2d:TableMap. Every r2d:ColumnBridge must specify a value for either this construct or the r2d:belongsToBlankNode construct. Example: The “Name” predicate in Figure 2 is associated with the resource “URI/StudentA”, an instance of the “Student” r2d:TableMap. Hence, the BTTM construct corresponding to “Name” r2d:ColumnBridge is set to a value of “Student”, thereby connecting the ColumnBridge to a table.

r2d:belongsToBlankNode (BTBN): This construct ties a r2d:ColumnBridge or MVCB to an SVBN or an MVBN. Example: The “Street” r2d:ColumnBridge corresponding to the “Street” predicate in Figure 2 is associated with the “Address” SVBN. Hence, for the “Street” r2d:ColumnBridge the BTBN construct is used to associate it to the “Address” blank node.

r2d:refersToTableMap (RTTM): This construct is optional for column bridges and is only used for those triples that contain a resource object for a predicate. This construct is used to generate primary key-foreign key relationships within the virtual relational schema.

Example: The object of the “Member Of” predicate in Figure 2 is a resource that translates to another r2d:TableMap called “Department”. Hence the “MemberOf” r2d:ColumnBridge includes the RTTM construct with a value of “Department”.

r2d:Predicate: The r2d:Predicate construct is used to store the fully qualified property name of the predicate which corresponds to the column bridge. This information is used during the SQL-to-SPARQL translation to generate the SPARQL WHERE clauses required to obtain the value of the r2d:ColumnBridge.

r2d:MultiValuedPredicate (MVP): This construct is used when there are multiple predicate names that refer to the same overall object type despite each individual object having a different value. r2d:MultiValuedPredicates are also used to keep track of the predicates associated with RDF containers and RDF collections.

r2d:Datatype: This construct specifies the datatype of its column bridge and is derived from the rdfs:range predicate or, in its absence, by examination of the object values of the predicate.

The virtual relational schema generated by R2D for the scenario in Figure 2 is as illustrated in Figure 3. Section IV (B) explains how this schema is arrived at.

![Figure 3. Equivalent Relational Schema for Scenario in Figure 2](image)

IV. R2D: A PROTOTYPE DESIGN

In addition to the design of the RDF-to-Relational mapping language discussed in the previous section, the objectives of this research are to develop algorithms that enable the relationalization of RDF stores. These algorithms comprise the R2D framework and are discussed in the following subsections.

A. RDFMapFileGenerator

The RDFMapFileGenerator algorithm automatically generates an RDF-to-Relational mapping file. It takes as input the RDF Store that is to be transformed and produces the transformation mapping file as output. The RDFMapFileGenerator algorithm works on RDF Stores with or without structural/schema information.

When structural information about the triples database is present the RDFMapFileGenerator algorithm discovers the
schema definitions and creates appropriate Table and Column mappings based on the schema information. Predicates belonging to instances with structural information are processed and added to the r2d:tableMap corresponding to the "rdf:class" of the instance using the constructs defined in Section III.

Instances without structural information are handled by creating a potential TableMap for each such instance. For every simple predicate of such resources, a new column is added to the resource's TableMap if a column corresponding to a predicate does not already exist in the TableMap. Additionally, the nature of the relationship that exists between all predicates (both pre-defined and undefined) and the subject is also determined. If the subject contains multiple object values for the same predicate then column type of the corresponding column is set to MVB. Otherwise, the column type is set to r2d:ColumnBridge. This determination is mandatory in order to arrive at a normalized and logically sound relational schema.

Furthermore, cardinality estimation is performed during the processing of predicates for those predicates that link subjects to objects that are resources and not literals. Whenever 1:N or N:M relationships are identified the corresponding predicate is mapped using the MVB or MVB construct, whichever is applicable. Once all predicates are processed, the potential TableMap is compared with other existing TableMaps; if an identical TableMap exists, the potential TableMap is discarded, otherwise it is added to the list of TableMaps.

B. DBSchemaGenerator

The DBSchemaGenerator module is the next stage in the R2D process. This algorithm takes the RDF-to-Relational Schema mapping file generated in Section III (A) and returns a virtual, appropriately normalized relational database schema consisting of entities/tables and the relationships between them. A high-level description of the algorithmic details follows.

For every entry of type r2d:TableMap in the map file one relational table is added to the virtual relational schema. For the sample scenario in Figure 2, a virtual table called Student is created corresponding to the Student r2d:TableMap. The more complex structures such as r2d:SingleValueBlankNodes (SVBNs), r2d:MultiValueBlankNodes (MVBs), and r2d:MultiValueColumnBridges (MVCBs) are handled as follows. For SVBNs, the predicates belonging to the blank node are associated with the table corresponding to the subject of the blank node object. Thus, the Street, City, and State predicates of the Address SVBN in Figure 2 are added as columns to the Student table.

When an MVB or MVB with literal objects is encountered (this is equivalent to a multi-valued attribute in relational database terminology) a new table is added to the virtual relational schema and the primary key fields of the table associated with the r2d:belongsToTableMap construct specified for the MVBN or MVCB are added as fields to this new table.

The object of the "Phone" predicate in Figure 2 is an example of an MVBN. The relational transformation for Phone involves the generation of an r2d:TableMap of the same name. This Phone r2d:TableMap includes as columns a Type field that holds the values of the multi-valued predicates off of the MVBN (in our sample scenario, the Type field will hold the values "Cell" and "Work"), and a Value field that holds the object values of the predicates off of the MVBN. Additionally, the r2d:TableMap also includes, as foreign key, the Student_PK column which references the primary key of the Student r2d:TableMap.

In RDF graphs where the MVBN or MVCB has objects that are resources themselves (as indicated by the r2d:refersToTableMap construct specified for the MVBN or MVCB), the type of relationship that exists between the subject and the object of the MVBN/MVCB is assessed. If an N:M relationship exists between the {subject, object} pair, a join table is added to the virtual relational table list and the primary key fields of both the tables (corresponding to the subject and the object) are added to this join table. The Works_On predicate in Figure 2 is an example of one such MVCB whose relational transformation results in the generation of a new r2d:TableMap of the same name. This new TableMap represents the N:M relationship between Student and Research and has the primary keys of both the tables included as fields. If the {subject, object} pair shares a 1:N or N:1 relationship, the primary key of the referred table is added to the attribute list of the referring table.

Finally, entries of type r2d:ColumnBridge in the map file are processed by adding the column bridge as an attribute to the table or blank node referred to in the r2d:belongsToTableMap or r2d:belongsToBlankNode construct specified for the column bridge.

C. ProcessSQLStatement

The final stage in the R2D process is the SQL-to-SPARQL translation where SQL statements issued against the virtual relational schema are parsed, translated into equivalent SPARQL queries that are executed against the RDF Store, and the results are returned in relational format. The algorithm for this stage is called ProcessSQLStatement. Due to space constraints only a brief description of this algorithm is provided here. Very broadly, for every field in the original SQL select list, a variable is added to the SPARQL SELECT list. Next, the predicates of every non-primary-key field in the SPARQL SELECT list are retrieved and added to the SPARQL WHERE clause to bind the SELECT list variables. SQL WHERE clauses of the types (field <operator> <value/field2>) are typically included in the FILTER clause which is then added to the SPARQL Query. The transformed SPARQL Query is executed, and the retrieved data is returned to the visualization tool in relational format seamlessly. Figure 4 shows a sample SQL...
query and its SPARQL equivalent and tabular results as generated by ProcessSQLStatement.

![SQL-to-SPARQL Transformation](image)

**Figure 4: SQL-to-SPARQL Transformation**

V. IMPLEMENTATION SPECIFICS

The hardware used in the implementation of R2D was a computer running Windows Vista with 2 GB RAM and 2.00 GHz Intel Core2 Duo Processor. The software tools used include Jena 2.5.6 to manipulate the RDF triples, MySQL 5.0 to house the relational equivalent of the given RDF store, Java 1.5 for development of the algorithms detailed in Section IV, and DataVision 1.2.0 to visualize/generate reports based on RDF data. The performance experiments conducted and the reporting tool outputs presented below are based on the IngentaConnect’s publication domain’s information about journals, issues, and articles. Synthetic RDF triples data stores of various sizes were created based on IngentaConnect’s schema in Jena for performance evaluation exercises.

The relational equivalent of the RDF data set was generated using the RDFMapFileGenerator and DBSchemaGenerator Algorithms detailed in Section IV. An open source visualization tool, DataVision, which expects a relational schema as input, was used to view the virtual relational schema generated, query the data using SQL statements, and generate reports off of the data. Figure 5 displays the time taken by the map file generation process for RDF stores of various sizes and the database schema as seen by DataVision.

![Map file generation time](image)

**Figure 5: Response Time for RDFMapFileGenerator**

Figure 6 illustrates DataVision’s query building process. Based on the fields chosen (in the “Report Designer” window), the table linkages (i.e., joins, illustrated in the “Table Linker” inset) specified, and additional record selection criteria specified (illustrated in the “Record Selection Criteria” inset), DataVision generates an appropriate SQL query, as shown in the “SQL Query” inset, to extract the required data. At this juncture, the Statement, PreparedStatement, and ResultSet JDBC Interfaces are invoked which trigger ProcessSQLStatement and return the results to DataVision in the expected tabular format.

![DataVision Query Processing](image)

**Figure 6: DataVision Query Processing**

The map file generation process is especially time-intensive for large databases without structural information (which is the case with our experimental data set) since RDFMapFileGenerator has to explore every resource to ensure that no property is left unprocessed. Sampling methods can be used to improve performance, but at the risk of reduction in accuracy. Also, if a domain expert is available, this step can be bypassed completely by providing a map file manually.

---

1 http://jena.sourceforge.net/index.html
2 http://datavision.sourceforge.net
3 www.ingentaconnect.com
To compare the performance of queries executed through the virtual relational schema offered by R2D against the query performance from an equivalent RDBMS, a physical relational schema corresponding to the publications data was created in MySQL and populated with data similar to the triples data in the Journal-Issue-Article RDF data store in Jena. Four queries were run against Jena RDF stores and MySQL relational databases of various sizes and the response times are displayed in Figure 7.

![Figure 7: Response times for Queries](image)

The time taken for the SQL-to-SPARQL conversion (ProcessSQLStatement Algorithm) is negligible and nearly constant. Hence, R2D does not add any overheads to the SPARQL query performance. The fact that the relational (SQL) queries exhibit superior performance than their SPARQL equivalents is not surprising since refined performance optimization options have been at the disposal of relational databases for many decades now. Further, for each row of the RDBMS with ‘n’ columns, there are ‘n’ triple tuples in the corresponding RDF Store. Thus, for the datasets considered, the RDBMS equivalent of the RDF Stores had approximately two-thirds less data than the RDF Stores which was another contributor to better RDBMS response times than the RDF data store.

However, this improved performance comes at the expense of additional disk space due to duplication of data into the RDBMS, and additional system resources/human effort required to ensure that the duplicated data is kept synchronized with the original RDF store. On the other hand, for a small price in terms of response time, R2D offers an avenue for users to continue to take advantage of the vast assortment of visualization tools that are readily available without having to duplicate/synchronize RDF data.

VI. DISCUSSION

The R2D framework in this paper is an attempt at integrating relational concepts with semantic web concepts with the objective of permitting reusability of tools that are based on a relational model. Since current storage methods for RDF stores involve housing the triples in a relational database, some factions may consider R2D to be a "double-wrapping" application that provides a relational wrapper around RDF stores that are, in turn, stored in a relational database. However, almost every storage mechanism involves the creation of a generic, non-application-specific <s,p,o> table that would make the determination of the problem domain addressed by the model difficult without examining the actual data. Further, querying data, using SQL, from such a generic table, to arrive at meaningful information is not a trivial task. It would involve umpteen self-joins on the same table and would require the presence of a domain expert with detailed knowledge of the data. This is because, using these models, it would be impossible for a user to infer the schema and the entities, the attributes, and relationships comprising the same. R2D offers the users the ability to do just this and enables them to actually arrive at a complete domain-specific Entity-Relationship Diagram using the RDF-to-Relational Schema transformation process and fire SQL queries against the same.

Further, R2D, unlike other mapping efforts, can generate an equivalent relational schema even for "sloppy" data (in which ontological constructs/schema definitions are absent) through extensive examination of the data to identify groups of instances that have mostly the same properties associated with them. The degree of accuracy of the generated schema in the absence of structural information may not be as high as when such information is available due to uncertainties regarding similarity of the tables generated in the relational schema. Decisions such as "how similar should two tables be before they are considered to be the same and consolidated" depends, in the absence of structural information, on similarity thresholds set within the algorithm and accuracy varies depending on the thresholds.

REFERENCES