Milan: Automatic Generation of R2RML Mappings

Sahil Nakul Mathur*, Declan O’Sullivan, and Rob Brennan

ADAPT Centre, School of Computer Science and Statistics,
Trinity College, Dublin, Ireland
mathurs@tcd.ie,{declan.osullivan,rob.brennan}@cs.tcd.ie

Abstract. Milan automatically generates R2RML mappings between a source relational database and a target ontology. It uses a novel multi-level algorithm to address the inter-model semantic gap by resolving naming conflicts and structural or semantic heterogeneity. This enables high fidelity mapping generation for realistic databases that are denormalised or utilise features of the relational data model that do not easily map to RDF. Milan is unlike many state of the art mapping systems which first produce a direct mapping ontology, and then apply ontology alignment techniques. Despite the importance of mappings for interoperability across relational databases and ontologies, a labour and expertise-intensive task, the current state of the art has achieved only limited automation. An experimental evaluation of Milan with respect to the state of the art systems using the Relational-to-Ontology Data Integration (RODI) metric is provided which shows that Milan outperforms all systems in all categories.

Keywords: RDB2RDF, OBDA, Schema and Ontology Matching, Mapping Rules, Linked Data, Automatic Mapping

1 Introduction

The Semantic Web promises easy data integration but in practice, this depends on the availability of detailed mappings, especially when converting from other data models like the relational model to RDFs data model [14]. Schema and ontology matching research [14] provide tools and methods to identify semantic correspondences between models such as database schema and OWL ontologies or Linked Data vocabularies. These correspondences are then validated, usually by domain experts, and then formalised as mappings which can be expressed in standard languages such as the W3C R2RML (Relational to RDF Mapping Language) recommendation [1]. However creating mappings is hard; it requires labour, domain expertise, and knowledge modeling expertise. Hence automated approaches are extensively studied. Relational database (RDB) to RDF mappings have additional complexity since the relational and RDF data models

* Corresponding Author

[1] www.w3.org/TR/r2rml/
do not exactly align, have different expressivity and each emphasizes a different modeling repertoire [11]. Hence for human validation of RDB to RDF mappings, additional expertise is required in both RDB and R2RML/RDF modeling. This, coupled with the prevalence of relational data in the enterprise mean the potential gains of automating relational to RDF mapping generation are considerable.

To date, most Relational database (RDB) to RDF mappings have additional complexity since the relational and RDF data models do not exactly align, have different expressivity and each emphasizes a different modeling repertoire [13]. Hence for human validation of RDB to RDF mappings.

The research question studied in this paper is To what extent can an automatic RDB to RDF mapping generation technique, Milan, based on semantic, lexical and structural analysis of both a source relational database and a target ontology create complete and accurate R2RML mappings?

Milan creates correspondences based on heuristic RDB to RDF mapping patterns we have observed in real databases. It uses a multi-level algorithm to detect class-table, object property-referential integrity and data property-column correspondences for realistic, complex relational databases. In order to minimise human intervention, these correspondences use Levenstein distance-based fuzzy label matching and identify optimal matches using combinatorial optimization. R2RML mappings are then generated to express these correspondences.

The contributions of this paper are as follows: (1) Milan, a novel method for detection of correspondences between relational databases and semantic web ontologies or vocabularies; and (2) an evaluation of the performance Milan against the Relational-to-Ontology Data Integration (RODI) benchmark [11] and comparison to state of the art mapping generation techniques.

The contributions of this paper are as follows: (1) Milan, a novel method for detection of correspondences between relational databases and semantic web ontologies or vocabularies; and (2) an evaluation of the performance Milan against the Relational-to-Ontology Data Integration (RODI) benchmark [11] and comparison to state of the art mapping generation techniques.

This paper is structured as follows: section 2 describes the use cases for relational-to-ontology mapping generation, section 3 outlines the requirements that a mapping generation system must address, section 4 describes the state of the art mapping generation systems and discusses how they address the stated requirements, section 5 details the multi-level algorithm of Milan, section 6 describes and analyses the performance of Milan against state of the art systems and finally section 7 presents the contributions and discusses the future scope.

2 Use Cases: RDB to RDF Mapping Patterns

In order to correctly detect mappings, it is important to be able to bridge between the different methods of expressing concepts in relational and ontology models (sometimes called the impedance mismatch over the object-relational gap [11]). Practically this is the ability to match the observed patterns of relational databases to the structure of ontologies. For example, a well-studied
pattern is to match classes to tables, object properties to primary-foreign key relationship and datatype properties to non key columns. This works well with simple databases. However, relational databases are designed and maintained with varied information models based on application-specific functional requirements. Many databases are not planned with known future requirements and evolve over time, which can lead to de-normalization, redundancy, and anomalies [5]. Thus real-world databases often diverge from the idealized use of the relational model and this creates additional complexity when identifying correspondences between relational and RDF models. Sequeda et al. have already identified a set of common mapping patterns between RDB and RDF models [13]. Their work is extended here by identifying new use-cases or mapping patterns (Fig. 1) which must be covered to generate high precision mappings between real relational databases and ontologies. The new patterns were identified by examining real datasets like the Norwegian Petroleum Directorates FactPages (NPD) which is a realistically large and complex database with a well-defined set of target ontology features in the RODI benchmark reference datasets. A table in a database is analogous to a class in ontology. When a single table is uniformly transformed into a class we call it a 1:1 (simple) mapping. But often a single table can correspond to multiple classes in an ontology and vice versa. Label matching often cannot detect these cases as label similarity occurs at the table to superclass level or to only one of a set of appropriate target sibling classes. Case (a) and (b) respectively of Fig. 1 shows how a single column can contain categorical values which should split a table into multiple fragments to match a set of subclasses or a set of siblings. The NPD dataset shows that these complex mappings can be more prevalent than simple mappings: while

![Fig. 1. Complex matching patterns found between databases (left) and ontologies (right). C: is for classes and T: is for tables. The thick-double arrowed line represents similarity across labels, the thin dotted line non-explicit relationships while the thin un-dotted line represents explicit relationships. The patterns observed are (a) and (b) Table split by column value to sub-classes and siblings respectively, (c) Relationship of 1:1 table with sub-classes, (d) Junction tables and object properties (e) Label collisions between data properties and columns](image-url)

2 https://data.norge.no/nlod/en/2.0

3 70 tables, 250,000 records and target ontology with 344 classes, 148 object properties, and 237 data properties
only 39 1:1 simple class to table mappings exist, 45 complex mappings fall under case (a) or case (b). For example, NPD Table `pipeline` and RODI target class `pipeline` has a column `pipMedium` which has `Oil` and `Gas` as categorical variables. The class `pipeline` represents this by having 2 sub-classes `OilPipeline` and `GasPipeline`. Instead of splitting table by columns, Fig. 1 case (c) shows how databases sometimes include 1:1 relationships between two tables where the central table label is related to the parent class of a set of classes that correspond to the 1:1 tables. This is a form of a de-normalized database. The child tables only contain a single column which is both its primary and foreign key. While the label of the parent table and parent class would match, its sub-classes and the child tables, matched by label matching, will not be able to match the properties of the sub-classes and the child table. In the NPD table `cmt` renamed (discussed later) the majority (23 of 69) foreign-key primary key relationships are this 1:1 denormalised form. Junction tables are associative entities that are used to implement the many-to-many relationship in information models stored in relational databases. Fig. 1 (d) shows this pattern. Table `T_1` : `{label_a}` is related to another table `T_2` : `{label_b}` via a junction table `JT` such that `JT`'s primary key is a composed foreign key of `T_1` : `{label_a}` and `T_2` : `{label_b}`. If a corresponding class pair is `C_1` : `{labelA}` and `C_2` : `{labelB}`, object properties across the class pair should be matched to the missing referential link between `T_1` : `{label_a}` and `T_2` : `{label_b}`. This poses a challenge to match these object properties to a true but non-existent link between the tables. In `cmt renamed` 30 out of 69 foreign-key primary key relationships are of this kind. When using label matching to identify property correspondences there are often name collisions. This is due to the fact that database administrators often designate column labels that are very similar. Case (e) of Fig. 1 is an example of how columns `la_i` and `la_v` would have equal likelihood to match with datatype property `La`. However, for the case of datatype properties, the range of the datatype in the database and target ontology is usually specified and can be a source of mapping context. In addition to the above patterns, naming heterogeneities such as tokenization based on naming conventions, token re-ordering and strings which are not words in languages can pose serious challenges in the discovery of mappings. For example, `label_ab_cd ≈ labelAbCd, abLabelCd ≈ cd_label_ab` and `label ≈ label` under many string matchers.

3 Requirements

The following requirements have been derived based on examination of the RDB to RDF use cases described above and the data integration classification as described by [1] i.e. naming disambiguation, structural and semantic heterogeneity. In order to accurately generate RDB to RDF mappings an automatic system must be able to:

**R1:** Generate RDB to RDF mappings an automatic system must be able to generate mappings (e.g. a R2RML file) given a source RDB and an OWL2, RDFS or Linked Data vocabulary enriched target ontology as inputs with high accuracy
and low levels of human intervention required. In particular, the automated system should not require additional human annotation of the source or target designed to support the mapping generation process.

**R2:** Address naming disambiguity across labels of classes, tables, keys, object properties, datatype properties and columns by: **R2a:** recognizing naming convention, tokenization, and token re-ordering. **R2a:** Be able to resolve data property-column label matching collisions as both columns and properties have very similar labels.

**R3:** Address structural heterogeneity by resolving type conflicts resulting from different modeling of class hierarchies [5], normalization, de-normalization. **R3a:** It should be able to identify 1:1 de-normalized table-table relationships, 1 : 1, 1 : n, n : 1 and m : n table-class, property-column relationships and n : m table-table relationships using junction table. **R3b:** In the absence of primary key-foreign key relations, automatic systems should be able to detect these relationships and match them with object properties.

**R4:** Address semantic heterogeneity caused by impedance mismatch [13] over object-relational gap by accurately detecting and bridging the difference in semantic expressiveness and patterns between ontologies such as inverseOf, cardinality constraints and unionOf relations and relational databases such as class hierarchies using column splitting and 1:1 table relationships.

### 4 Related Work

While a number of semi-automatic RDB to RDF mapping generation systems exist, this paper focuses on fully automatic systems and so limits the scope of this section accordingly. Automatic systems can be divided into one-stage and two-stage systems. Two-stage systems such as BootOx[7], Automap4OBDA[15] produce a target-agnostic ontology followed by ontology alignment to the target ontology. One stage systems such as IncMap[10] directly map without the use of intermediate ontology. Other inter-model mapping tools include -ontop-[8], MIRROR[8], D2RQ[2] and COMA++[4]. Table 1 summarises whether each of the systems address particular requirements mentioned in Sec. 3.

BootOx[7] applies direct mapping and provides support for different OWL profiles. It enriches the bootstrapped ontology using explicit and implicit database constraints from the RDB that creates axioms about the classes and properties. This is followed by ontology alignment using LogMap. BootOx is best at resolving semantic heterogeneity by using OWL features. [6].

IncMap[10] exploits mapping patterns to enrich their graph-based data structure representing RDB and RDF data model. It then uses lexical and structural analysis to obtain correspondences between RDB and ontology. However, IncMap addressed limited mapping patterns and doesn’t focus on semantic heterogeneity.

Automap4OBDA (A4MO) [15] is based on RDBToOnto [12]. Unlike IncMap, A4MO uses ontology learning techniques to extends correspondences between

---

[1] www.w3.org/TR/owl-ref
[2] www.w3.org/TR/rdb-direct-mapping/
Table 1. Comparison of state of the art systems with respect to requirements (Req.)

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Req.</th>
<th>B.OX</th>
<th>IncM.</th>
<th>Ontop</th>
<th>MIRR.</th>
<th>COMA</th>
<th>D2RQ</th>
<th>A4MO</th>
</tr>
</thead>
<tbody>
<tr>
<td>Prop. Inheritance</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Col. Split</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
<tr>
<td>Inv.(Obj. Prop)</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
<td>✓</td>
</tr>
</tbody>
</table>

the data and classes of target ontology. While both BootOx and A4MO enrich their bootstrapped/putative ontology, only A4MO addresses class hierarchies. However, A4MO is not as versatile as IncMap in inferring class hierarchies. A4MO is better than other systems in addressing structural heterogeneity.

The following lists the gaps in the current state of the art systems. They all fail to detect $n : m$ due to junction table and $n : 1$ class-table relationships due to column splitting. They also miss out on many properties matching for tables that have 1 : 1 relationship with other tables. They don’t utilize annotation properties, resolve datatype property-column label collisions. They also underperform at addressing semantic heterogeneity. They don’t use the expressivity of OWL such as InverseOf, unionOf to enrich the relational-ontology relationship.

5 Automatic Mapping Generation Algorithm

Milan has been developed to fill the gaps identified by the requirements and in the state of the art mapping generation systems. Milan uses multi-level algorithms that leverage heuristic matching patterns to discover mappings between a relational database and target ontology.

Milan comprises of three main correspondence identification processes and 6 supporting processes that are invoked by the main processes as needed (Fig. 2). The source relational database and target ontology act as inputs to Milan and it produces a set of R2RML mappings as an output.

The Class-Table Relationship (Sec. 5.3) process detects 1 : 1, $n : 1$, and some 1 : $n$ class-table relationships (Req. R3a) using Label Matching (Sec. 5.1). Column Splitting and Combinatorial Optimization (Sec. 5.2) Column Splitting detects categorical values in columns, thereby splitting the table based on these values. These values are matched to sub-classes and sibling classes (Req. R3a,R4). The class-table matching result is input to the Object Property Foreign Key/Primary Key process to discover links using Referential Integrity (Req.
3,4), Label Matching (Req. 2), Junction Table Detection and Combinatorial Optimization. Referential Integrity discovers the keys linking two pairs of matching class-tables and the object properties across the two classes. Junction Table Detection enriches existing relationships with the inclusion of implicit many-to-many relations across two or more tables (Req. 3).

These relations are then matched to object property by Referential Integrity process. In addition to this, the main process also detects 1:1 table mappings, where sub-tables inherit columns and keys from its parent, thereby adding to link and attribute discovery. The results of class-table matching from the first and the inheritance of properties due to 1:1 table mappings from the second process then act as inputs to the Data Property-Column process to discover the relationship between datatype properties and non key columns for a given class-table pair (Req. 2,3,4). This process uses Label Matching, Datatype Partitioning, and Combinatorial Optimization to yield a match. Finally, Milan translates all correspondences from the three processes into R2RML mappings (Req. 1). Each of these functions is discussed in more detail below.

5.1 Label Matching

Milan uses a variant of FuzzyWuzzy[^6] uses Levenshtein Distance, producing scores in the range [0, 1]. Milan modified FuzzyWuzzy to add camelCase and special character based tokenization. Label matching addresses the naming heterogeneity patterns involving token re-ordering using sort_token_ratio which re-orders the tokens based on lengths. Label matching uses a specific variant for Column Splitting (Sec. 5.3), where it filters out the superclass label token from its subclass or sibling label, thereby improving matching scores.

[^6]: [https://github.com/seatgeek/fuzzywuzzy](https://github.com/seatgeek/fuzzywuzzy)
5.2 Combinatorial Optimization

Milan uses the Hungarian algorithm [9] to solve assignment problem across class-table, data property-column and object property-FK/PK. For two set of labels \((x_m, y_n)\), it uses the label scores of \(x * y\) to obtain \(\min(m, n)\) 1:1 matches such that the sum of scores is the maximum.

5.3 Class-Table Relationship

The algorithm first detects 1:1 followed \(n : 1\) class-table relationships. Algorithm 1 takes class and table \(rdfs:label\) as input. It uses the fragment of the class URI in the absence of \(rdfs:label\). Label matching [5,1] is used over the two lists producing a three column matching table \(O_{a+b,3}\). \(O_{a+b,3}\) is then post-processed by filtering out results with low string match scores based on a threshold, which is determined experimentally \((\tau = 0.7)\). This is followed by filtering out lower match scores due to duplicate occurrences of any class or table label, represented as \(uniqueCriteria\) in Algorithm 1.

The result of this is a one-one match table. The next step is to address the patterns of Fig.[1][case (a) and (b)]. Each 1:1 class-table pair is considered for column splitting by considering unique elements of all non-key columns. This is done by gathering all non-key columns \((W_{m,1})\) along with sub-classes and sibling
classes($V_{n,1}$) for the given table-class pair. Then, for each column, the unique element values and their columns are stored in a two column table ($T_{a,2}$). In order to maximize computation efficiency and matching accuracy, columns having a very large number of unique elements are removed from consideration. This is done using a threshold which is experimentally determined to be $\Theta = 1.2 \times n$. A modified variant of Label Matching is then run over this list of unique values with sub-classes and siblings of a target class against all labels of unique column values, using score threshold $\tau$. Data diversity is enhanced by the overlooking label of target class when considering the sub/sibling class labels. Using the Hungarian algorithm over the resultant label match table provides 1:1 column value-class matches, which are optimized by maximizing the sum of scores $5.2$.

5.4 Object Property-Referential Integrity Relationship

Algorithm 2 uses rdfs:domain, rdfs:range, owl:UnionOf and owl:inverseOf of object property $\theta$ (Alg. 2.3) along with referential integrity (FK/PK relationship) $R$ using primary-foreign key pairs represented by foreign column names $R$ (Alg. 2.2). Using the class-table pair from Sec. 5.3, it obtains the corresponding two pairs of classes and tables. Algorithm 2 matches the obtained list of object properties with keys of the relational database using label matching. However, a naive approach will miss out on detecting complex relationships $3$ such as junction tables, one-to-one table relations, and inverse relationships. Alg.2 detects junction tables via pattern mentioned in 1 and infers a many-to-many relationship across the table pair. The label used to describe this relationship is the junction table’s name. Alg.2 also detects 1:1 table-table mappings and inherits all columns of parent table to the child tables. Lastly, while relationships in relational databases have a clear sense of directionality, relationship across the class may not always be uni-directional. Sometimes they also have a pair of properties which are inverse of each other. This is represented using rdfs:InverseOf predicate. Milan overcomes this impedance mismatch by dropping directionality (Alg.2.6) and prioritizing direction (Alg.2.10) in the presence of alternates. Table 2 lists the challenges that Alg.2 resolves by reasoning over rdfs:InverseOf. Consider case(a) of Table 2 both object property present (OP) and FK/PK relation (Col) are present for relation $C_1/T_1 \rightarrow C_2/T_2$ (1). However in relation (2), the rdfs:InverseOf(A), $X$ is present, but FK/RK relation is not present. Hence $a$ has

<table>
<thead>
<tr>
<th>Class/Table Direction</th>
<th>OP</th>
<th>Col</th>
<th>OP</th>
<th>Col</th>
</tr>
</thead>
<tbody>
<tr>
<td>(1) $C_1/T_1 \rightarrow C_2/T_2$</td>
<td>$A$</td>
<td>$a$</td>
<td>$A$</td>
<td>$a$</td>
</tr>
<tr>
<td>(2) $C_2/T_2 \rightarrow C_1/T_1$</td>
<td>(X) inv:A</td>
<td>$a$</td>
<td>(X) inv:B</td>
<td>$b,a$</td>
</tr>
</tbody>
</table>
to be inferred, which will be matched to \( X \). In case (b), single object property and FK/PK relations are present in opposite direction. The FK/PK relation is therefore inferred to match the object property. Case (c) is a specific case of junction table relationships, where more than one relation exists across two tables. It could occur that both the object property \( A \) & \( B \) have inverses. And corresponding FK/PK relations are divided across different directions. Inferences are made to include missing FK/PK relations for each relation. The final matching is performed by label matching and then performing Hungarian algorithm with \( \text{labelMatch}(\text{Property}, FK/PK_a) \gg \text{labelMatch}(\text{inv: Property}, FK/PK_b) \) an additional constraint for every property and its inverse pair. This is to prevent a property and its inverse, both to occur as a result of the Hungarian algorithm.

**Algorithm 2: Object Property and Referential Integrity**

1. Procedure: opRefInt\( (R, O, N) \)
2. \( R \leftarrow \) Referential Constraints of RDB
3. \( O \leftarrow \) Object Property and rdfs:InverseOf relations
4. \( N \leftarrow \) Class-Table relationship from Algorithm 1
5. \( R \leftarrow R \cup \text{TransferObjectProp}(\text{DetectTableOneToOne}(R)) \)
6. \( R \leftarrow R \cup \text{Inverted}(R) \) with inversion annotation
7. \( R \leftarrow R \cup \text{DetectJunctionTables}(R) \)
8. \( L_c \) & \( L_T \leftarrow R \cup O \) using \( N \) in class pairs (c) & table pair format (T)
9. For each \( i \) in \( L \)
10. \( P \leftarrow L_c[i] \)
11. \( C \leftarrow L_T[i] + \text{InferredColumns}(L_T[i], L_c[i]) \)
12. \( P, C \leftarrow \text{FilterPriority}(P, C) \) where \( DR > 1DR \)
13. \( M_{i,3} \leftarrow \text{fuzzyWuzzy}(P, C) \)
14. \( N \leftarrow \text{HungarianAlgorithm} \_ \text{constraint}(M_{i,3}) \)
15. Return \( N \)

The sequence of Alg. 2 is different from the sequence in which this section described it. Algorithm 2, above presents the exact sequence of operations.

### 5.5 Datatype Property and Column Relationship

The Datatype of data property is retrieved by querying its rdfs:label, rdfs:range and if present its owl:UnionOf. Milan performs label matching across all data properties and nonkey columns for each class-table pair. In addition to this, Milan addresses label collisions by segregating datatypes based on W3C datatype mapping. The rationale behind this segregation is likes match. i.e a column having varchar datatype cannot possibly match to a datatype property having xsd:date as its range. Hence offering greater accuracy in cases of string ambiguity. For a given pair of class-table, each segregation of a datatype group e.g Varchar – xsd:string, Milan uses Hungarian algorithm to obtain a 1:1 match based on label matching scores. It is important to note that this algorithm is also capable of detecting 1 : n relationship for a property-column pair. This is done in three ways. One, when multiple properties related to each other by owl:sameAs. Second, owl:UnionOf contains multiple classes for a given datatype property.

\footnote{www.w3.org/TR/r2rml/}
hence mapping the same property to multiple tables. Lastly, when detecting

1:1 table-table relationships in [5,4] child tables inherit all columns of the main

table, allowing possible matches to data property.

5.6 R2RML Mapping Generation

Many papers such as [13] have discussed R2RML mapping patterns. In addition
to those, Milan has additional features to support 1:1 table inheritance and 1:n
table-class relationship based on column splitting. The SQL queries involve a
JOIN condition. These query templates are trivial hence is left to readers.

6 Evaluation

The purpose of the evaluation is to test the quality of mappings generated by
Milan in realistic deployment scenarios. Several papers such as Tarasowa et al
[18] and RODI benchmark [16] and RODI benchmark [11] have discussed the
quality evaluation aspects of mappings. The evaluation in this paper uses the
RODI benchmark and methodology.

The RODI suite [8] includes a wide range of relational-to-ontology scenarios,
each based on mapping challenges similar to Sec. [3] Each scenario provides an
input database and a target ontology. It requests from systems a complete set of
mappings that enable to execute queries over the target ontology (T-Box). Each
scenario contains a list of SPARQL-SQL query pairs, which are categorized under
various mapping challenges such as class-table, attributes, links. These query
pairs are executed to evaluate if the results from the SQL queries match the
SPARQL queries to the ontology constructed using the mappings provided. It
then calculates the averages of precision and recall of all queries in each scenario,
thereby calculating a fitness function for mapping quality. In addition to total
scores, RODI also provides aggregated scores for each category of the query
pairs.

8 https://github.com/chrpin/rodi
This paper evaluates Milan based on 4 scenarios provided by RODI. The RODI authors have already performed ontology alignment of output ontologies with the target ontology for systems like MIRROR and -ontop-, which don't produce mappings. They have also provided mapping translation for systems like D2QRQ and COMA++, by translating system specific mapping language to R2RML. Thus our approach has allowed Milan to be evaluated against a wide variety of mapping generation systems that have already been evaluated against RODI. It is important to note that this RODI-based evaluation uses 4 scenarios, two of which were not examined for mapping pattern identification or algorithm development for Milan and hence demonstrate the wider applicability of the Milan’s algorithms.

Below we describe the hypothesis of our RODI experiment on Milan, the scenarios used for evaluation, the results and finally a discussion of the analysis that can be inferred from the results.

**Hypothesis** The hypothesis is that Milan’s approach performs better than the current state of the art in terms of the mapping quality metric used by RODI.

**RODI Benchmark** This section describes the four scenarios used to evaluate Milan. Table 3 provides a description of the scenarios. Each scenario has varied difficulty levels. For instance, the relational schemata of cmt_renamed closely follows modeling patterns from their corresponding ontologies. cmt_structured additionally introduces 1:n class hierarchy mapping challenge (R3a,4). conference_no FK is void of primary key-foreign key relations making it tough to detect object property mappings (R3b). npd_atomic, which is based on the NPD dataset is the most challenging scenario of all. 1:n matches (R3a,4), for both classes and properties are present. 1:n matches as a structural feature can therefore best be tested in the npd_atomic_tests scenario [11]. It also contains 17 SQL-SPARQL queries where each require a significant number of schema elements to be correctly mapped at the same time to bear any results. These queries represents that automatic systems should be capable of addressing actual real-world queries accurately than any simplified query workloads [11].

<table>
<thead>
<tr>
<th>Scenario</th>
<th>Tables</th>
<th>FK/PK</th>
<th>Classes</th>
<th>Queries</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmt_renamed</td>
<td>48</td>
<td>69</td>
<td>23</td>
<td>30</td>
</tr>
<tr>
<td>cmt_structured</td>
<td>64</td>
<td>52</td>
<td>23</td>
<td>29</td>
</tr>
<tr>
<td>conference_noFK</td>
<td>60</td>
<td>0</td>
<td>59</td>
<td>39</td>
</tr>
<tr>
<td>npd_atomic</td>
<td>70</td>
<td>100</td>
<td>300</td>
<td>439</td>
</tr>
</tbody>
</table>
Results & Analysis

Table 4 shows RODI scores for each scenario on every tested system. Technically defined as per test F-measures, these scores indicate the percentage of successfully passed query tests. Although all systems are far from achieving complete accuracy in automatically generating mappings, Milan outperforms current state of the art systems. Therefore the hypothesis is accepted. Fig. 3 below is a breakdown of the aggregated score for cmt_renamed,

<table>
<thead>
<tr>
<th>Scenario</th>
<th>B.OX</th>
<th>IncM.</th>
<th>Ontop</th>
<th>MIRR.</th>
<th>COMA</th>
<th>D2RQ</th>
<th>A4MO</th>
<th>Milan</th>
</tr>
</thead>
<tbody>
<tr>
<td>cmt_renamed</td>
<td>0.76</td>
<td>0.66</td>
<td>0.28</td>
<td>0.28</td>
<td>0.48</td>
<td>0.31</td>
<td>0.56</td>
<td>0.86</td>
</tr>
<tr>
<td>cmt_structured</td>
<td>0.41</td>
<td>0.44</td>
<td>0.14</td>
<td>0.17</td>
<td>0.38</td>
<td>0.31</td>
<td>0.41</td>
<td>0.52</td>
</tr>
<tr>
<td>conference_noFK</td>
<td>0.33</td>
<td>0.41</td>
<td>-</td>
<td>0.17</td>
<td>0.21</td>
<td>0.18</td>
<td>0.41</td>
<td>0.46</td>
</tr>
<tr>
<td>npd_atomic</td>
<td>0.14</td>
<td>0.16</td>
<td>0.10</td>
<td>0</td>
<td>0.02</td>
<td>0.08</td>
<td>0.23</td>
<td>0.30</td>
</tr>
</tbody>
</table>

where Milan performs better across queries under class, data property and object property. Milan correctly detected 75/134 class-table, 42/213 data properties-

![Score Break-down for cmt_renamed on classes, data property & object property](image)

The impact of Algorithm 1 is evident in the npd_atomic dataset. It correctly detects 56% of the total class-table mappings. The outcome of this algorithm also detected 12 out of 45 of the 1 : n class-table match is present. Although cmt_structured had many patterns reflecting column splitting, Algorithm 1 was not so successful here. 6 columns were split based on binary values, e.g TRUE of column is_Reviewer of table Person would match to Reviewer class. Additionally, there are cases of column splits which detected categorical values such “1” and ”2”, which were either foreign or non-key column. These were true matches to separate classes but label matching failed to match these cases. Presently, Milan only looks into immediate sibling or sub classes. However cmt_structured show undetected correspondence where the matching class was a sub-sub-class.

The relatively high scores achieved in the object property-referential integrity relationship are credited to junction table detection, 1:1 relationship matching followed up property inheritance and managing inverse property relations. It also
detected 1:1 table-table relationships, which comprises of 34% of the primary-foreign key relations. This increased object property mappings due to inheritance. However, Milan couldn’t capture object property mappings when object properties don’t have explicit rdfs:range/domain or owl:unionOf.

Better results with detecting data property are credited to the use of datatype partitioning and the Hungarian algorithm with Label Matching for the class-table pair. In addition to this, the algorithm uses annotation properties such as rdfs:label and rdfs:comment to match to the column. This is the only cause for a marginal improvement (12%) in scores in conference_nofk scenario. Additionally, the performance is enhanced by the inheritance of properties due to 1:1 table-table detection. However, the strategy of datatype partitioning had its compromises. The tables of conference_nofk scenario has columns which have boolean datatype while the matching data property is a string. Milan is unable to match these cases, thereby missing on some of the data properties.

7 Conclusion

Milan has been demonstrated to automatically generate R2RML mappings between a relational database and target ontology with an overall f-measure of between 0.86 and 0.3 under the RODI benchmark conditions. This measure is an indicator of the accuracy and completeness of the mappings generated. In this evaluation, Milan has out-performed all the other automatic mapping generation systems that have been tested to date with RODI. The relative performance of Milan improves in more complex scenarios like npd_atomic, where it performed 30.4% better than the next best system, A4MO. Similarly, the performance pf Milan was 26.8% better than the next leading system, IncMap, in the cml_structured scenario. This demonstrates the relative effectiveness of a direct mapping generation approach, as implemented by Milan, compared to the more popular putative ontology-based mapping generation methods.

Further work is required to broaden our confidence in this work by evaluating Milan over additional datasets. In addition this evaluation has helped us to identify the following potential RDB-RDF mapping patterns or improvements to Milan: Detecting class-table mappings (Req. 3a) where multiple tables form one class in the target ontology; extending the column-based table splitting pattern to account account for foreign key columns (Req. 4) ; and inferring implicit links between tables based on undeclared foreign keys whose use is observed in the database (Req 3b).

Acknowledgments. This research has received funding from the ADAPT Centre for Digital Content Technology, funded under the SFI Research Centres Programme (Grant 13/RC/2106) and co-funded by the European Regional Development Fund.
References

12. Šeleng, M., Laclavík, M., Balogh, Z., Hluchý, L.: Rdb2onto: Approach for creating semantic metadata from relational database data