# Milan: Automatic Generation of R2RML Mappings

Sahil Nakul Mathur[1,3], Declan O'Sullivan[1], and Rob Brennan[1,2]

[1] ADAPT Centre, School of Computer Science and Statistics, Trinity College, Dublin
mathurs@tcd.ie, {declan.osullivan,rob.brennan}@cs.tcd.ie
[2] School of Computing, Dublin City University, Dublin
[3] Facebook Ireland, Dublin

**Abstract.** *Milan* automatically generates R2RML mappings between a source relational database and a target ontology, using a novel multi-level algorithms. It address real world inter-model semantic gap by resolving naming conflicts, structural and semantic heterogeneity, thus enabling high fidelity mapping generation for realistic databases. 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. The paper describes an experimental evaluation of *Milan* with respect to the state of the art systems using the RODI benchmarking tool 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 [13]. Schema and ontology matching research [13] 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 [4]. 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 do not exactly align, have different expressivity and each emphasizes a different modeling repertoire [10], [12]. Hence for human validation of RDB to RDF mappings, additional expertise is required in both RDB and R2RML/RDF modeling.

[4] www.w3.org/TR/r2rml/

This, in the enterprise context means the potential gains of automating relational to RDF mapping generation are considerable.

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 [10] and comparison to state of the art mapping generation techniques.

## 2 Related Work

This section briefly discusses different state of the art systems. Pinkel et al [10] have performed a detailed state of the art review of selected automated systems. 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[6], Automap4OBDA[14] produce a target-agnostic ontology followed by ontology alignmnent to the target ontology. One stage systems such as IncMap[9] directly map without the use of intermediate ontology. Other inter-model mapping tools include -ontop-[3],MIRROR[7], D2RQ[2] and COMA++[4].

BootOx [6] applies direct mapping [5] 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. [5].

IncMap [9] 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) [14] is based on RDBToOnto [11].Unlike IncMap, A4MO uses ontology learning techniques to extends correspondences between the data and classes of target ontology. While both BootOx and A4MO enrich

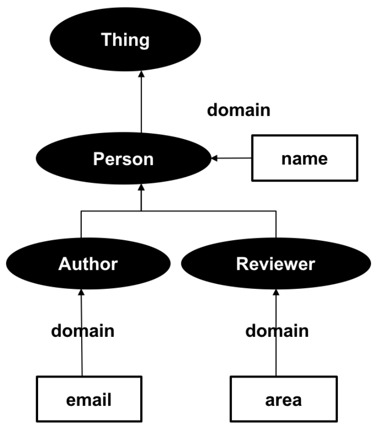
[5] www.w3.org/TR/rdb-direct-mapping/

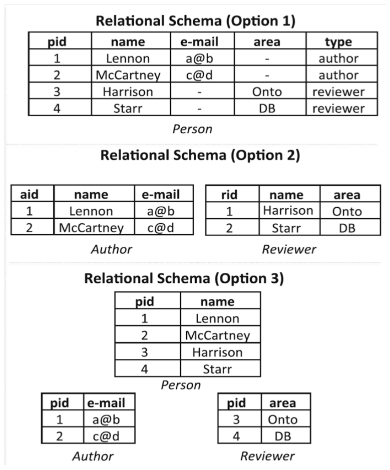
their bootstrapped/putative ontology, only A4MO addresses class hierarchies. However, A4MO is not as versatile as IncMap in inferring class hierarchies. A4MO is better than other systems in addressing structural heterogeneity.

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 under-perform at addressing semantic heterogeneity. They don't use the expressivity of OWL such as *InverseOf*, *unionOf* to enrich the relational-ontology relationship.

## 3 Requirements

As mentioned above, most real-world relational schema and corresponding ontologies cannot be related using nave direct mappings as there are vast differences in the ways which same concepts are modelled. This section references examples of different data modelling techniques discussed in state of the art. Based on such observations, literature also provides a comprehensive set of requirements for automatic mapping systems. These are briefly discussed below.





**Relational Schema (Option 1)**

| pid | name | e-mail | area | type |
|---|---|---|---|---|
| 1 | Lennon | a@b | - | author |
| 2 | McCartney | c@d | - | author |
| 3 | Harrison | - | Onto | reviewer |
| 4 | Starr | - | DB | reviewer |

*Person*

**Relational Schema (Option 2)**

| aid | name | e-mail |
|---|---|---|
| 1 | Lennon | a@b |
| 2 | McCartney | c@d |

*Author*

| rid | name | area |
|---|---|---|
| 1 | Harrison | Onto |
| 2 | Starr | DB |

*Reviewer*

**Relational Schema (Option 3)**

| pid | name |
|---|---|
| 1 | Lennon |
| 2 | McCartney |
| 3 | Harrison |
| 4 | Starr |

*Person*

| pid | e-mail |
|---|---|
| 1 | a@b |
| 2 | c@d |

*Author*

| pid | area |
|---|---|
| 3 | Onto |
| 4 | DB |

*Reviewer*

**Fig. 1.** Examples demonstrating different modelling repertoire

C.Pinkel et al[10] discusses various modelling repertoire using tables such as author and reviewer. This is shown in Figure 1 above. A popular set of classifications for mapping challenges in relational-to-ontology patterns, which was first discussed by Batini et al [1] and subsequently discussed by Pinkel et al [10] , are as follows: (1) naming conflicts, (2) structural heterogeneity and (3) semantic heterogeneity. Table 1, which is adopted from [10] summarizes the requirements of relational-to-ontology mapping approaches for automated systems. C.Pinkel et al[10] discusses each of these requirements in detail.

**Table 1.** Requirements for addressing relational-to-ontology mapping challenges

| ID | Challenge Type | Relational-to-ontology systems requirement |
|---|---|---|
| **R1** | **Input-Output** | *System should accept relational database and ontology as inputs and return mappings/mapping, correspondences as output* |
| | **NAMING CONFLICTS** | |
| **NC1** | **Tokenization** | *Matching label via different tokenization conventions* |
| **NC2** | **Datatype Disambiguation** | *Distinguishing, datatype property column labels based on datatypes. (e.g. xsd:date cannot,match to a integer label)* |
| | **STRUCTURAL HETEROGENITY** | |
| **ST1** | **Normalization** | *Weak entity table* |
| **ST2** | | *1:n attribute,(class:table)* |
| **ST3** | | *1:n relation (class:table)* |
| **ST4** | | *n:m relation (class:table)* |
| **ST5** | **De-normalization** | *Correlated entities* |
| **ST6** | | *Multi-values* |
| **ST7** | **Class Hierarchies** | *1:n property-column match with type column* |
| **ST8** | | *n:1class with type column* |
| **ST9** | | *n:1 class with type column* |
| **ST10** | **Key Conflicts** | *Plain composite key* |
| **ST11** | | *Missing keys* |
| **ST12** | | *Missing reference* |
| **ST13** | **Dependency Conflicts** | *1:n attribute* |
| **ST14** | | *1:n relation* |
| **ST15** | | *n:m relation represented by junction table* |
| | **SEMANTIC HETEROGENITY** | |
| **SE1** | **Impedance Mis-match** | *OWL profile support. E.g Inverse properties* |

## 4 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* comprises of three main 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.

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 reorders the tokens based on lengths. Label matching uses

[6] `https://github.com/seatgeek/fuzzywuzzy`

a specific variant for Column Splitting (Sec. 4.1), where it filters out the superclass label token from its subclass or sibling label, thereby improving matching scores. Combinatorial Optimization: *Milan* uses the Hungarian algorithm [8] to

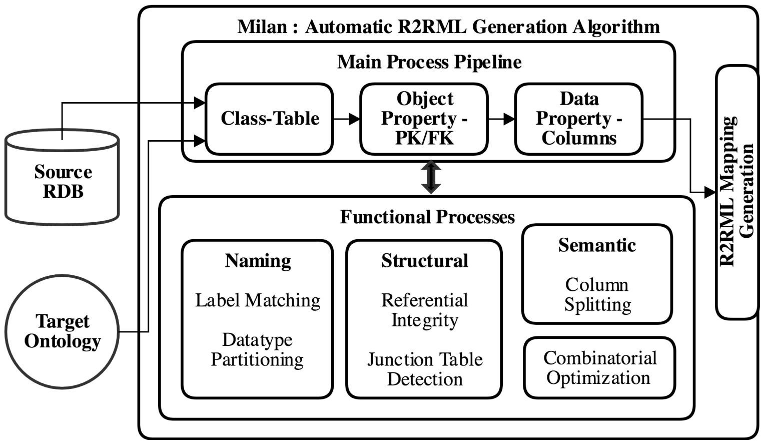


**Fig. 2.** Architecture of *Milan*

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.

## 4.1 Class-Table Relationship

The Class-Table Relationship (Sec. 4.1) process detects $1 : 1$, $n : 1$, and some $1 : n$ class-table relationships using Label Matching, Column Splitting and Combinatorial Optimization. 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.

The class-table matching result is input to the Object Property Foreign Key/Primary Key process to discover links using Referential Integrity, Label Matching, 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.

The algorithm first detects 1:1 followed $n : 1$ class-table relationships. Algorithm 1 takes class and table *rdfs:label* as input. Label matching 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. 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 $\Theta = 1.2 * n$. 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.

**Algorithm 1 : Class – Table Relationship**

```
1:  Procedure: getClassTableMain (X,Y)
2:  $X_{a,1}$← List of all Classes
3:  $Y_{b,1}$← List of all Tables
4:  $O_{a*b,3}$← labelMatching(X,Y)
5:  $O_{n,3}$ ← $O_{a*b,3}$ where $O_{a*b,3}[,3] > \tau$ & uniqueCriteria
6:  if(n < ζ)
7:          For each i in n
8:                  $M_{a,3}$ ← subClassDetection($O_{n,3}[i,1]$, $O_{n,3}[i,2]$)
9:                  $N_{*+a,3}$ ← V.concat($N_{*,3}$ , $M_{a,3}$)
10: $N_{p,3}$← V.concat($N_{p,3}$, $O_{n,3}$)
11: Return $N_{p,3}$
```

**Algorithm 1a : Detecting N:1 Relationships**

```
1:  Procedure: subClassDetection(class, table);
2:  $X_{1,1}$← Class Name
3:  $Y_{1,1}$← Table Name
4:  $V_{n,1}$← Get all Sub-classes and Siblings($X_{1,1}$)
5:  $W_{m,1}$← Get all non key Columns of Table($Y_{1,1}$)
6:  Initialize ColumnVals[,2]
6:  For each i : m
7:      If count.unique($W_{m,1}[i]$) < θ
8:          $T_{a,2}[,1]$← unique($W_{m,1}[i]$) ; $T_{a,2}[,2]$← $W_{m,1}[i]$
9:          $U_{*+a,2}$ ← V.concat($V_{*,2}$, $T_{a,2}$)
10: $S_{a,3}$ ← LabelMatching_modified ($V_{n,1}$, $U_{b,2}[,1]$)
11: $M_{\min(b,n),3}$ ← HungarianAlgorithm($S_{a,3}$)
```

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.

## 4.2 Object Property-Referential Integrity Relationship

Algorithm 2 uses *rdfs:domain*, *rdfs:range*, *owl:UnionOf* and *owl:inverseOf* of object property *0*, Alg. 2,3 along with 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. 4.1, 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 naïve 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 patterns 3 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.

**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:  $R \leftarrow R \cup$ TransferObjectProp(DetectTableOneToOne($R$))
6:  $R \leftarrow R \cup$ Inverted($R$) with inversion annotation
7:  $R \leftarrow R \cup$ DetectJunctionTables($R$)
8:  $L_c$ & $L_T \leftarrow R \cup O$ using $N$ in class pairs (c) & table pair format (T)
7:  For each i in L
8:          $P \leftarrow L_c[i]$
9:          $C \leftarrow L_T[i]$ + InferredColumns($L_T[i], L_c[i]$)
10:         $P, C \leftarrow$ FilterPriority($P, C$) where $DR > iDR$
11:         $M_{n,3} \leftarrow$ fuzzyWuzzy($P, C$)
12:         $N \leftarrow$ HungarianAlgorithm_constraint($M_{n,3}$)
13: Return $N$
```

**Table 2.** Inferring columns for object property (OP) matching to accomodate impedance mismatch due to *rdfs:InverseOf*. Inferred foreign columns (Col) in bold

| Class/Table Direction | OP | Col |
|---|---|---|
| (1) $C_1/T_1 \rightarrow C_2/T_2$ | A | a |
| (2) $C_2/T_2 \rightarrow C_1/T_1$ | (X) *inv:A* | ***a*** |

(a)

| OP | Col |
|---|---|
|  | a |
| A | ***a*** |

(b)

| OP | Col |
|---|---|
| A,B | a,***b*** |
| (X) *Inv:A*, (Y) *inv:B* | b,***a*** |

(c)

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 $\boldsymbol{a}$ has to be inferred, which will be matched to $X$. In case(b)2, 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)2 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 $labelMatch(Property, FK/PK_a) \gg labelMatch(inv : Property, FK/PK_b)$ 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.

### 4.3 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[7]. 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 *ow:sameAs*. Second, *owl:UnionOf* contains multiple classes for a given datatype property, hence mapping the same property to multiple tables. Lastly, when detecting

**Algorithm 3 : Datatype Property and Columns**

```
1:  Procedure: DpC(class, table);
2:  $X_{n,1}$← List all classes
3:  $Y_{m,1}$← List all tables
6:  For each i in X
7:          For each j in Y
8:                  $D_{a,2}$← Datatype_Annotation Property_&Range($X_{n,1}[i]$)
9:                  $C_{b,2}$← Non_Key_Columns_&Datatype($Y_{m,1}[j]$)
10:                 For each k in (a ∪ b ) using RDB – RDF datatype mapping
11:                   $M_{n,3}$ ← fuzzyWuzzy($D[,1], C[,1]$)
                      where $D[,2] = C[,2] = k$
12:                   $N$ ← HungarianAlgorithm($M_{n,3}$)
13: Return All $N$
```

1:1 table-table relationships in 4.2, child tables inherit all columns of the main table, allowing possible matches to data property.

[7] www.w3.org/TR/r2rml/

### 4.4 R2RML Mapping Generation

Many papers such as [12] 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.

## 5 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 [15] and RODI benchmark [10] 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.

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 dont produce mappings. They have also provided mapping translation for systems like D2QRQ and COMA++, by translating system specific mapping language to R2RML. 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.

**Benchmark Datasets** This section describes the four scenarios used to evaluate *Milan*. These scenarios are based on real open data such as Conference, Norwegian Petroleum Directorate [10].

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_nofk* 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

[8] `https://github.com/chrpin/rodi`

**Table 3.** Description of the RODI scenarios used for evaluation. *Tables* and *FK/PK* represent number of tables and primary key-foreign key relations in the source RDB.*Classes* represents the number of classes in the target ontology. *Queries* provides the number of SQL-SPARQL queries executed for evaluation

| **Scenario** | **Tables** | **FK/PK** | **Classes** | **Queries** |
|---|---|---|---|---|
| *cmt_renamed* | 48 | 69 | 23 | 30 |
| *cmt_structured* | 64 | 52 | 23 | 29 |
| *conference_nofk* | 60 | 0 | 59 | 39 |
| *npd_atomic* | 70 | 100 | 300 | 439 |

(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 [10].The queries are designed to check on the existence of accurate mappings [10].

**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. *Milan* outperforms current state of the art systems. Therefore the hypothesis is accepted. Fig. 3 below is a

**Table 4.** Comparison of overall scores based per test F-measure using RODI

| **Scenario** | **B.OX** | **IncM.** | **Ontop** | **MIRR.** | **COMA** | **D2RQ** | **A4MO** | **Milan** |
|---|---|---|---|---|---|---|---|---|
| *cmt_renamed* | 0.76 | 0.66 | 0.28 | 0.28 | 0.48 | 0.31 | 0.56 | **0.86** |
| *cmt_structured* | 0.41 | 0.44 | 0.14 | 0.17 | 0.38 | 0.31 | 0.41 | **0.52** |
| *conference_nofk* | 0.33 | 0.41 | - | 0.17 | 0.21 | 0.18 | 0.41 | **0.46** |
| *npd_atomic* | 0.14 | 0.16 | 0.10 | 0 | 0.02 | 0.08 | 0.23 | **0.30** |

breakdown of the agregated score for *cmt_renamed*, where *Milan* performs better across queries under class, data property and object property.

*Milan* correctly detected 75/134 class-table, 42/213 data properties-column and 15/92 object property-FK/PK matches in the *npd_atomic*.

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. 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.

The relatively high scores achieved in the object property-referential integrity relationship are credited to junction table detection, 1:1 relationship matching followed by property inheritance and inverse property inferencing. It also detected 1:1 table-table relationships, which comprises of 34% of the primary-foreign key relations. This increased object property mappings, caused due to

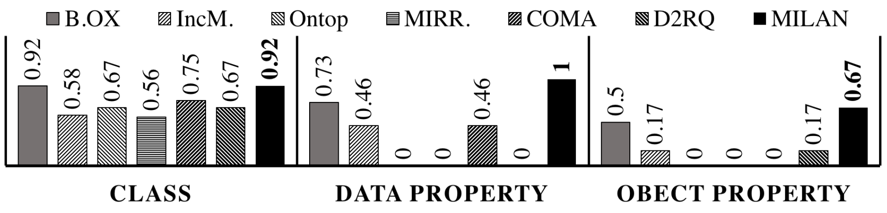


**Fig. 3.** Score break-down for cmt_renamed on classes, data property & object property

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. 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.

## 6 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. *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 *cmt_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 where multiple tables form one class in the target ontology; extending the column-based table splitting pattern to account account for foreign key columns ; and inferring implicit links between tables based on undeclared foreign keys whose use is observed in the database.

**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

1. Batini, C., Lenzerini, M., Navathe, S.B.: A comparative analysis of methodologies for database schema integration. ACM computing surveys 18(4), 323–364 (1986)
2. Bizer, C.: D2rq - treating non-RDF databases as virtual RDF graphs. In: In Proceedings of the 3rd International Semantic Web Conference ISWC (2004)
3. Calvanese, D., Cogrel, B., Komla-Ebri, S., Kontchakov, R., Lanti, D., Rezk, M., Rodriguez-Muro, M., Xiao, G.: Ontop: Answering SPARQL queries over relational databases. Semantic Web 8(3), 471–487 (Jan 2017)
4. Do, H.H., Rahm, E.: COMA: A System for Flexible Combination of Schema Matching Approaches. In: Proceedings of the 28th International Conference on Very Large Data Bases. pp. 610–621. VLDB '02, VLDB Endowment, Hong Kong, China (2002)
5. Jiménez-Ruiz, E., Grau, B.C.: Logmap: Logic-based and scalable ontology matching. In: International Semantic Web Conference. pp. 273–288. Springer (2011)
6. Jimnez-Ruiz, E., Kharlamov, E., Zheleznyakov, D., Horrocks, I., Pinkel, C., Skjveland, M.G., Thorstensen, E., Mora, J.: BootOX: Practical Mapping of RDBs to OWL 2. In: The Semantic Web - ISWC 2015. pp. 113–132. Lecture Notes in Computer Science, Springer, Cham (Oct 2015)
7. Medeiros, L.F.d., Priyatna, F., Corcho, O.: MIRROR: Automatic R2rml Mapping Generation from Relational Databases. In: Engineering the Web in the Big Data Era. pp. 326–343. Lecture Notes in Computer Science, Springer, Cham (Jun 2015)
8. Munkres, J.: Algorithms for the Assignment and Transportation Problems. Journal of the Society for Industrial and Applied Mathematics 5(1), 32–38 (Mar 1957)
9. Pinkel, C., Binnig, C., Jimenez-Ruiz, E., Kharlamov, E., Nikolov, A., Schwarte, A., Heupel, C., Kraska, T.: IncMap: A Journey towards Ontology-based Data Integration. Gesellschaft fr Informatik, Bonn (2017)
10. Pinkel, C., Binnig, C., Jimnez-Ruiz, E., Kharlamov, E., May, W., Nikolov, A., Sasa Bastinos, A., Skjveland, M.G., Solimando, A., Taheriyan, M., Heupel, C., Horrocks, I.: RODI: Benchmarking relational-to-ontology mapping generation quality. Semantic Web 9(1), 25–52 (Jan 2018)
11. Šeleng, M., Laclavík, M., Balogh, Z., Hluchỳ, L.: Rdb2onto: Approach for creating semantic metadata from relational database data
12. Sequeda, J., Priyatna, F., Villazn-Terrazas, B.: Relational Database to RDF Mapping Patterns. In: Proceedings of the 3rd International Conference on Ontology Patterns - Volume 929. pp. 97–108. CEUR-WS.org, Germany (2012)
13. Shvaiko, P., Euzenat, J.: Ontology Matching: State of the Art and Future Challenges. IEEE Transactions on Knowledge and Data Engineering 25(1), 158–176 (2013)
14. Sicilia, ., Nemirovski, G.: AutoMap4obda: Automated Generation of R2rml Mappings for OBDA. In: Knowledge Engineering and Knowledge Management. pp. 577–592. Lecture Notes in Computer Science, Springer, Cham (Nov 2016)
15. Tarasowa, D., Lange, C., Auer, S.: Measuring the quality of relational-to-rdf mappings. In: International Conference on Knowledge Engineering and the Semantic Web. pp. 210–224. Springer (2015)