Automatic Generation of Relational to Ontology Mapping Correspondences

A thesis submitted to the

University of Dublin, Trinity College

in fulfilment of the requirements of the degree of

Master in Science (Research) Computer Science

Sahil Nakul Mathur

Knowledge and Data Engineering Group (KDEG)

School of Computer Science and Statistics

Trinity College

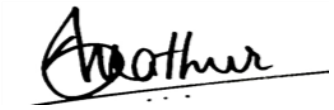
Supervised by Dr. Rob Brennan

Co-supervised by Prof. Declan O'Sullivan

2019

Declaration

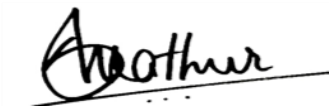
I, the undersigned, declare that this work has not been previously submitted as an exercise for a degree at this or any other University, and that, unless otherwise stated, it is entirely my own work.



Sahil Nakul Mathur

Permission to lend or copy

I, the undersigned, agree that the Trinity College Library may lend or copy this thesis upon request. I consent to the examiner retaining a copy of the thesis beyond the examining period, should they so wish.



Sahil Nakul Mathur

Table of Contents

List of Figures

List of Tables

Acknowledgements

I would like to thank my supervisors, Dr. Rob Brennan and Professor Declan O'Sullivan, for all of their support and guidance throughout this MSc. Their patience, motivation, and immense knowledge helped me in all the time of research and writing of this thesis. I am thankful for an ecosystem where my inquisition was addressed with utmost patience, paving way to self-discovery of my questions. The ADAPT Centre was an intellectually stimulating, highly collaborative environment. The peer-to-peer learning enabled me to undergo the journey of research much better. My thanks and best wishes to all my friends and colleagues at ADAPT Centre.

Besides the wonderful Rob and Declan, my sincere thanks to Prof. Dinesh Singh, former Vice Chancellor of University of Delhi who mentored me to think out of the box, to take risk. He reiterated Mahatma Gandhi's urge to use "your hands", which metaphorically meant to follow a path of scientific experimentation and discovery. Thank you for infusing those values, that passion and a sense of goal in life.

My thanks to Dr. Gyana Parija and Dr. Amit Nanavati, under whom I worked at IBM Research before joining MSc at Trinity College. I am blessed to have received continued encouragement, guidance and faith during the MSc.

Thankyou Dr. Soumyabrata Dev for your suggestions on the thesis, your company in the good times and your support during the tough days. Thanks Arunima Sethi and Anuj Singh, who made Trinity College and Dublin even more beautiful. Each of you took great effort in reviewing and advising on matters pertaining to the thesis.

Lastly, my sincere thanks to my parents, Mukul and Sanjeela and my brother Pratyaksh for their unwavering support. The thesis is dedicated to my late grandfather Krishna C. Sinha, himself a scholar. I lost him in the duration of this thesis.

Abbreviations

CSV	Comma Separated Values
OWL	Web Ontology Language
RDF	Resource Description Framework
RDFS	RDF Schema
SPARQL	SPARQL Protocol and RDF Query Language
SQL	Structured Query Language
DM	Direct Mappings
RDB	Relational Databases

Abstract

This thesis presents Milan, an automatic relational-to-ontology system. Milan automatically generates mapping correspondences from a source relational database (RDB) and a target ontology. It addresses the relational-to-ontology challenges by resolving naming conflicts, structural and semantic heterogeneity. This enables high fidelity mapping correspondence generation for realistic databases that are de-normalised or utilize features of the relational data model that do not trivially map to RDF.

Through a systematic review of the state-of-the-art in relational-to-ontology patterns and automatic relational-to-ontology mapping correspondence systems, the thesis sums up requirements automatic systems should address and also identifies the gaps present in the current state-of-the-art. The thesis leverages this analysis to design Milan, constituting three main processes and six sub-processes. The description of each process is done using UML process diagrams, algorithms and input/output templates. The thesis uses a popularly used benchmarking tool to evaluate Milan and other state-of-the-art systems. The evaluation experiment involves scenarios and tests, which not just describe the overall performance of a system but also quantitatively describes the various mapping challenges where the system fairs well and poorly.

This research is supported by Science Foundation Ireland through the ADAPT Centre (www.adaptcentre.ie) at Trinity College Dublin.

1. Introduction

1.1. Motivation

The Semantic Web promises easy data integration, but in practice this depends on the availability of mapping correspondences, especially when converting from other data models like the relational model to the Resource Description Framework (RDF) data model [1]. Schema and ontology matching research [1] provides tools and methods to identify mapping correspondences [2] between models such as database schema and OWL ontologies or Linked Data vocabularies. Mapping correspondences state how the entities and relationships of relational databases are encoded in the various concepts of the ontology. These correspondences are formalised as mappings [3] which can be expressed in standard languages such as the W3C R2RML (Relational to RDF Mapping Language) recommendation [4].

However creating mapping correspondences is hard; it requires labour, domain expertise, and knowledge modelling expertise [2]. Hence automated approaches have been and continue to be extensively studied. Relational-to-ontology mapping correspondences involve complexity since the relational and RDF data models do not exactly align, have different expressivity and each emphasizes a different modelling repertoire [3]. This mismatch is sometimes called impedance mismatch [2]. This fact coupled with the prevalence of relational data in the enterprise mean the potential gains of automating relational to RDF mapping generation are considerable. The well-studied pattern for relational-to-ontology mapping correspondences is to match tables to classes, primary-foreign key relationships to object properties and non-key columns to datatype properties [2]. This works well with simple databases and ontologies. However, relational databases (RDB) are often normalized in such a manner that cannot be addressed by simply modelling a table to a class. Relational databases (RDB) are designed and maintained with varied information models predicated on application-specific functional requirements. Many databases are extemporary without knowledge of future requirements and they also evolve over time, which can lead to de-normalization, redundancy, and anomalies [6]. Thus, real-world databases such as

the Norwegian Petroleum Directorate[1] (NPD dataset) often diverge from the idealized use of the relational model and this creates additional complexity when identifying correspondences or mappings between relational and RDF models.

More automatic mapping generation systems capable of generating complete and accurate RDB to RDF mapping correspondences can thus potentially result in more efficient data integration for large and complex relational databases, e.g. for ontology based data access (OBDA) ([7],[8]) to relational databases, or result in a reduction in manual effort during the mapping creation process.

Accurate and complete mappings mean that all features of the relational data model have been mapped to the corresponding features in the ontology model, such that, a query over a relational database will yield exactly the same result as a corresponding query over an ontology. This notion of accurate and complete mappings is formalised in Section 1.2 as quality of mapping correspondence.

Current RDB to RDF mapping generation approaches such as BootOx4 [9], IncMap [10], Ontop [11], MIRROR [12], COMA [13], D2RQ [14] and AutoMap4OBDA [15] are far from fully automatic mapping generation systems. For example, the best performance for fully automatic mapping creation found in our evaluation of such systems on a real dataset is 23%; indicating that only 23% of the total expected correspondences that exist are being automatically detected. The performance notion in the above context is based off accurate and complete mapping. Challenges in mapping generation have been classified as: naming disambiguity, structural and semantic heterogeneity [16].

Sequeda et al. have identified a set of common mapping patterns between RDB and RDF models [2]. However, current mapping generation systems are unable to detect these patterns in the relational database or ontology. Additionally, they do not fully utilize the expressivity of ontologies such as annotation properties and inverse property reasoning (`owl:InverseOf`). These gaps in the current state-of-the-art need to be addressed. Thus, it is argued in this thesis that an automatic mapping generation system is needed to meet the following requirements: bridge the inter

[1]www.data.norge.no/nlod/en/2.0

model gap by taking into consideration relational-to-ontology patterns; resolve naming ambiguities and address structural and semantic heterogeneity. It is to be noted that automatic mapping generation system and automatic mapping correspondence generation system mean the same in this thesis.

1.2. Research Question

The research question investigated in this thesis is:
To what extent can an automatic RDB to RDF mapping generation technique, based on semantic, lexical and structural analysis of both a source relational database and a target ontology result in quality mapping correspondences?

RODI benchmark provides datasets (referred to as scenarios) that can closely mimic the challenges encountered by mapping generation in the real world. Each input benchmark scenario (dataset), provided by RODI, contains a source relational database, a target ontology and sets of equivalent SQL and SPARQL test queries. Evaluation scenarios test (SQL-SPARQL queries) evaluates mappings for their accuracy and completeness. Hence the objective of the evaluation scenarios presented by Pinkel et al. [5], evaluate if all features of the relational data model have been mapped to the corresponding features in the ontology model, such that, real world queries over relational database will yield exactly the same result as a corresponding query over an ontology. The metric used to perform this evaluation is called Quality of mapping correspondences [5], which is the F-measure of the evaluation scenarios test (SQL-SPARQL queries). The following 3 research objectives have been identified from the research question:

- **RO1:** Analyse the state-of-the-art in relational-to-ontology mapping (RDB2RDF) patterns and automatic relational-to-ontology mapping generation systems;
- **RO2:** Design and develop a multi-level algorithm-based system that employs lexical, structural and semantic analysis to automatically discover correspondences between RDB and RDF;

- **RO3:** Evaluate the prototype over multiple datasets and compare results with results from other state-of-the-art mapping generation systems.

1.3. Contribution

The contribution is called Milan, which is a multi-level algorithm that performs lexical, structural and semantic analysis to create correspondences between RDB and RDF. Milan is based on heuristic RDB to RDF mapping patterns observed in real databases e.g. the NPD dataset mentioned earlier. It is envisaged that semantic web and database integration users and researchers will use Milan for mapping generation for data integration and to enable Ontology based Data Access (OBDA).

In addition to Milan's individual algorithms, re-usable artefacts arising from the research include; label matching, which is a modification of the FuzzyWuzzy[2]'s Levenshtein distance to incorporate naming conventions based tokenization; a modified Hungarian algorithm, which is an algorithm to solve assignment problems. It maximizes the sum of individual scores of Levenshtein distance (label matching).

Hungarian algorithm has been used in a novel way to resolve datatype collisions which matching data property to non-key columns; OWL inferencing, which utilizes owl expressivity to address impedance mismatch. Impedance mismatch refers to the conceptual difficulty to map graph representation of data to relational schemas, which are tabular and based on the relational algebra, which defines linked heterogeneous rows. For example, the case when multiple column pairs link two tables via a junction table and similarly multiple object properties link the corresponding two classes. Properties have a direction, from domain to range. The `owl:inverseOf` construct can be used to define an inverse relation between properties. Hence in order to discover RDB-to-RDF correspondences, a direction resolution has to be undertaken to match the column pairs of RDB to object properties of RDF. A column splitting method has been developed, which

[2] https://github.com/msubhash/fuzzywuzzy-java

considers each non-key column for column splitting by considering all unique elements.

1.4. Technical Approach

This research work has developed a multi-level algorithm to detect class-table, object property-referential integrity and data property-column correspondences. In order to minimise human intervention, the algorithms use Levenshtein distance-based fuzzy label matching and identify optimal matches using combinatorial optimization.

Milan comprises of three main correspondence identification processes and 6 supporting processes that are invoked by the main processes as needed (Fig.1). 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 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.

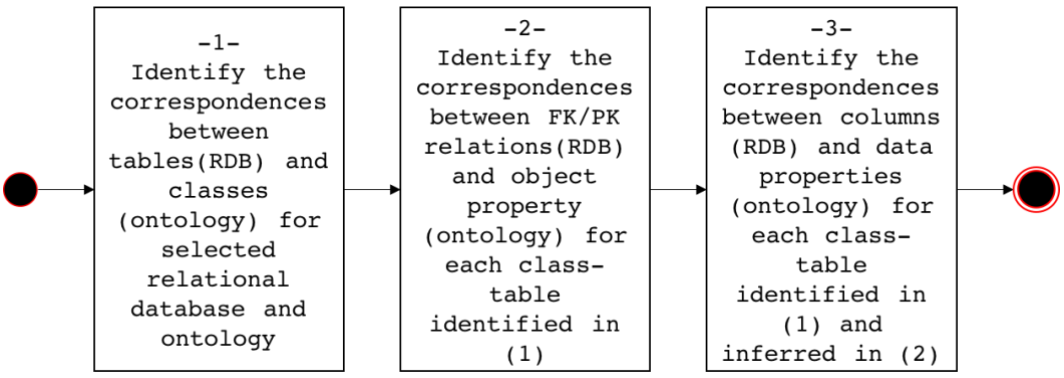


Figure 1 Milan main processes

The Class-Table matching process 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. These relations are then matched to object property by a Referential Integrity process. In addition to this, the main process also detects 1:1 table mapping, 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 process and the inheritance of properties due to 1:1 table mapping 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. This process uses Label Matching, Datatype Partitioning, and Combinatorial Optimization to yield a match. Finally, Milan uses the results to finally produce mapping correspondences from the three processes. In this thesis the mapping correspondences have been evaluated (see next section for more detail) for mapping quality using the RODI benchmark [5].

The multi-level algorithm has been implemented in this research using Java and the following packages: Jena[3], Hungarian Algorithm, FuzzyWuzzy. The source relational database has been hosted on PostgresSQL[4]. The target ontology has been hosted on Apache Fuseki[5]. Protégé[6] and DBVisualizer[7] have been used for schema visualization of RDF and RDB respectively.

1.5. Evaluation Methodology

Evaluations have been performed to test the quality of the mappings generated by Milan involving realistic database-ontology scenarios. The evaluation of Milan uses the RODI benchmark [5]. The selection of RODI was made as RODI offers

[3] www. jena.apache.org
[4] www.postgresql.org
[5] www.jena.apache.org/documentation/fuseki2
[6] www.protege.stanford.edu
[7] www.dbvis.com

comprehensive benchmarking, with real world datasets, single-globally comparable scoring measure for mapping quality. Additionally, the state-of-the art mapping systems that compare against Milan have been evaluated over RODI. Section 7 of Pinkel et al. [5] and the introduction of Chapter 5 discuss other related benchmarking systems, such as TPC benchmark [17] and Tarasowa et al. [18]. RODI benchmarking tool holistically evaluates automatic relational-to-ontology systems over varied mapping challenges. RODI has been extensively chosen as the evaluation methodology by many state-of-the-art relational-to-ontology mapping systems such as BootOx [9], IncMap [10], MIRROR [19] and Automap4OBDA [15].

RODI can accept customized scenarios for evaluation in addition to 12 pre-existing scenarios. Each scenario provides an input database and a target ontology. A complete set of mapping correspondences are the inputs, enabling RODI 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 and 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.

Milan has been evaluated using 4 of the scenarios provided by RODI. These scenarios have been selected based on criteria such as different domains, database size & complexity, variations in ontology modelling style, the presence of cardinality information and semantic expressivity of ontologies. Different domains allow evaluation of the lexical matching scope of Milan. Section 5.1.3 provide strong explanation on how these scenarios are representative enough to perform evaluation. Table 29 and Table 30 of Section 5.1.3 discuss these scenarios, the different mapping challenges present, and the extent to which they are present in each of the scenario. Future work in Section 6.3 indicates that Milan and other state-of-the-art mapping systems can be evaluated over more and latest scenarios.

Each scenario has varying difficulty levels. For instance, the relational schemas of *cmt_renamed* scenario closely follows modelling patterns from their corresponding ontologies. The *cmt_structured* scenario additionally introduces 1:n

class hierarchy mapping challenge. The *conference_nofk* scenario is void of primary key-foreign key relations making it tough to detect object property mappings. The *npd_atomic* scenario, which is based on the NPD dataset is the most challenging scenario of all. 1:n matches, 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. The scenario also contains 17 SQL-SPARQL queries where each requires a significant number of schema elements to be correctly mapped at the same time to bear any results. These queries are designed to show that automatic systems should be capable of addressing actual real-world queries accurately than any simplified query workloads. Each query pair test of a scenario is evaluated for accuracy and recall. The metric used is, per-scenario, f-measure, which is the average of precision and recall for all queries of a scenario [20].

As will be seen in Chapter 5, the evaluation shows the performance of Milan in addressing challenges that exist such as: attributes, which corresponds to data property – column correspondences; links, which corresponds to object property – primary-foreign key relationship. The results achieved by Milan for each of the challenges are compared to evaluation results achieved by a number of state-of-the-art systems.

1.6. Thesis Overview

The remainder of this thesis is structured as follows:

Chapter 2: Background

This chapter provides an insight into the information which is required to understand the background of ontology-based data integration and uplift of relational data to Linked Data. It discusses various components of relational database, which are relevant to the thesis. It describes the components that comprise linked data. It also focuses on the concept of uplift of data and ontology-based data access.

Chapter 3: State-of-the-art

This chapter presents the various relational-to-ontology mapping challenges. Requirements are then listed for automatic relational-to-ontology systems to address these challenges. It then describes BootOx, IncMap, MIRROR, COMA++, the seven state-of-the-art, automatic mapping systems, their capabilities are analysed and compared on the basis of these requirements. Lastly, gaps in the existing state-of-the-art are identified which motivates the development of Milan.

Chapter 4: Design

This chapter details the multi-level algorithm of Milan. Milan uses multi-level algorithms that relational-to-ontology matching patterns to discover mappings between a relational database and target ontology. It discusses how Milan has been developed to fill the gaps identified in the state-of-the-art mapping generation systems.

Chapter 5: Evaluation

This chapter describes and analyses the performance of Milan against state-of-the-art systems over four different datasets. The performance is tested using quality of mappings generated by Milan and other automatic mapping generation systems. The metrics used to describe quality is per-test f-measure. This chapter also analyses this performance for particular mapping challenges such as class-table correspondence, data property – column matching.

Chapter 6: Conclusion

Here, key findings are presented from the research described in this thesis. It discusses to what extent the research question of this thesis has been answered and the extent to which the research objectives have been met. Possible directions for future work related to the research in this thesis are also outlined.

2. Background

This chapter presents background information that will aid a reader that is unfamiliar with ontology-based data integration and uplift of relational data to Linked Data. Presented in Section 2.1 is an introduction to relational databases, their components and how they are stored and queried. Section 2.2 discusses Semantic Web ontologies, their data model, the components that comprise an ontology and how ontologies are stored and queried. Section 2.3 discusses the relational-to-ontology uplift approach, which includes ontology matching and mapping.

2.1. Relational Databases

Relational databases are based on the relational data model set forth by IBM Research's E. F. Codd [21]. The relational data model organizes data into one or more tables (relations) of columns (or fields) and rows (or records).

2.1.1. Components of a Relational Database

Tables, Rows, and Columns. A table is a collection of related information, which is organized by rows and columns. All rows in a table are based on the same set of columns. Each row in a table has its own unique value, also called the primary key. The key is used to uniquely identify a row in the table. It is important to note that a primary key comprises one or more (combination) columns in a table.

Foreign Key, this is a column value in one table that is referenced to match the column value of the primary key in another table. If the foreign key value is not null, then the primary key value in the referenced table must exist. It is this relationship of a column in one table to a column in another table that provides the relational database with its ability to join tables. The relationship between two tables can be classified as one-to-one, one-to-many (which is the most common type) or many-to-many.

A One-to-one (1:1) relationship exists when each row in one table has only one related row in a second table. i.e. the foreign key(s) of the table is also the primary key(s).

A one-to-many (1:n) relationship exists when each row in one table has one or many related rows in a second table. e.g. multiple departments can have the same manager.

A many-to-many (m:n) relationship exists when a row in one table has many related rows in a second table. Likewise, those related rows have many rows in the first table. This relationship cannot be represented merely by a primary key – foreign key relationship as the uniqueness of primary key cannot be maintained. This relationship can be converted into two one-to-many relationships by connecting these two tables with a junction table that contains the two related columns. The partial Entity-Relationship (ER) diagram of a relational database in Figure 2, demonstrates the use of junction tables. The junction table "READ_BY_REVIWER" allows a many-to-many relationship between the tables "PAPERS" and "REVIEWERS".

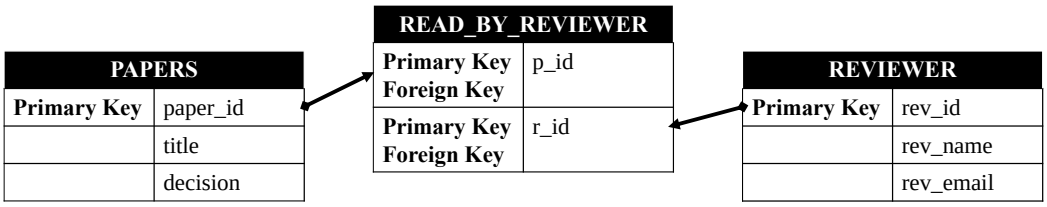


Figure 2: Example showing junction table "READ_BY_REVIEWER" establishes a many to many relationship between tables "PAPERS" and "REVIEWER"

These different foreign key-primary key relationships along with tables, rows, and columns form the key components of a relational database.

2.1.2. Hosting and Querying

A **Relational Database Management System (RDBMS)** is a database server system whose primary functions are to store data securely and return that data in response to requests from other software applications. Example database servers

include MySQL[8] and Postgres[9], each of which has additional features such as information schema tables which "provides access to database metadata, information about the MySQL server such as the name of a database or table, the data type of a column, or access privileges" [22]. In the context of relational-to-ontology mapping, it is important to note that the RDBMS information schema contains a list of foreign key-primary key relations and a list of columns with their datatypes for each table. The foreign key-primary key relations match to object properties in ontologies and the datatypes allow to narrow the search space for matching the non-key columns to data properties, which also contain a datatype information using `rdfs:range` property. These are further discussed in detail in the coming Section 2.3 and later in Section 4.4.3, which discussing relational-to-ontology patterns. An RDBMS provides a client application in the Java programming language through the standard Java Database Connectivity (JDBC)[10] interface via an application programming interface (API). This enables clients to access the functionality of the database server such as data querying, insertion, creation, and deletion.

Query answering using Structural Query Language (SQL), is based upon relational algebra. SQL is a set-based declarative programming language designed for data retrieval and manipulation. Figure 3 shows a sample SQL SELECT statement, which returns a result set of rows from one or more tables. SELECT statements have many clauses such as FROM; which specifies the table(s) from where the rows have to be retrieved, WHERE; which provides the condition to retrieve selected rows, NATURAL JOIN; which combines two or more tables based on foreign key-primary key relation, GROUP By is used with aggregate functions such as count to group the result-set by one or more columns.

[8] www.mysql.com
[9] www.postgresql.org
[10] http://www.oracle.com/technetwork/java/overview-141217.html

```
SELECT Paper.title AS Title,
       count(*) AS Authors
 FROM  Paper
 WHERE Paper.type = "Research"
 NATURAL JOIN  Paper_author
 GROUP BY Paper.title;
```

Figure 3: Sample SQL query

Section 2.1 presented a short description of relational databases. It covered the concept of tables, rows, different kinds of foreign key-primary key relationships. Additionally, it discussed RDBMS where databases can be hosted and queries using SQL and also accessed via an API in the Java programming environment using the JDBC.

2.2. Ontologies

An ontology is a "formal naming and definition of the types, properties, and interrelationships of the entities that really exist in a particular domain of discourse" [23]. Ontologies are analogous to relational models but intended for modelling knowledge about entities, their attributes, and their relationships to other entities [23]. Chiang et al. [24] provide analysis to distinguish data, knowledge, and information, where the authors claim that some consensus on the definition of knowledge is that, it can be considered data at a high level of abstraction and can be processed by a computer when it is represented as data. Ontologies, therefore, provide this abstraction. Due to this, ontologies are used for integrating heterogeneous databases and enabling interoperability. The Semantic Web [25] is an extension of the World Wide Web for distributed, web-scale ontologies defined through standards by the World Wide Web Consortium (W3C). According to the W3C, the Semantic Web[11] is about two things: (1) Its standards promote common

[11] www.w3.org/2001/sw/

data formats for heterogeneous data integration of data drawn from diverse sources, where on the original Web mainly concentrated on the interchange of human-oriented documents rather than machine-readable data [26] ; (2) It also defines modelling languages and protocols for recording how data relates to real-world objects, most fundamentally through the Resource Description Framework (RDF) [27] data model and semantics. Section 2.1.1 presents the components that comprise a semantic web ontology. The components can be classified into the data model and knowledge component. The data model includes the RDF triples, URI and Vocabularies. The knowledge component includes classes and properties. Section 2.2.2 then discusses how semantic web ontologies can be hosted on servers called triplestores and queries using SPARQL Protocol and RDF Query Language (SPARQL) [28].

2.2.1. Components of Semantic Web Ontologies

This section discusses the key components of a Semantic Web ontology. It starts with explaining the components of the RDF Data Model, which forms the fundamental basis of Semantic Web ontologies. The central element of an RDF data model is a resource, which has been defined in this section below. The relationship between these resources is described as an RDF triple. Resources are uniquely identified using a Universal Resource Identifier (URI) [29]. A Resource is the smallest unit in an RDF triple. The RDF data model is built upon the standard languages used for knowledge modelling. The second part of this chapter discusses aspects of knowledge modelling in ontologies. This sub-section contains a brief overview of vocabularies, individuals, classes, and properties. Vocabularies are used to provide semantic information about resources. For example a resource defining a class. Individuals linked to a class share some common features. Properties are resources that describe the relationship between two resources. Lastly, this section ends with an example of an ontology and a graphical visualization of the same.

Data Model

Resource. Are entities that can be anything including physical things, documents, abstract concept, numbers, string [29]. The term is synonymous with entities as it is described in RDF Semantics specification [30].

RDF triple. RDF makes statements about resources and how they are related to each other; these statements are in the form of subject-predicate-object expressions. A subject is a resource about which the triple is describing. Predicate defines the property or relationship of the subject resource. An object is the value of the property which could be a literal value or it can redirect to another resource [26]. Table 1depicts the role of subject, predicate, and object in the triple by providing a simple example.

Table 1: An example of an RDF Triples demonstrating the purpose of subject, predicate and object

<subject>	<predicate>	<object>
<Emma>	<is a>	<novel>
<Emma>	<was written by>	<Jane Austen>
<Jane Austen>	<is an>	<author>

URI stands for Universal Resource Identifier. A URI globally identifies a resource uniquely [29]. Hence every resource that exists in an RDF triple contains a URI. The URL used on the internet is also a form of URI. Table 2 rewrites the triples from Table 1 by replacing Jane Austen and Emma with URIs, which are DBpedia[12] resources. Text in yellow will be addressed in the next sub-section on vocabularies.

Table 2: Replacing some of the resources in the RDF triples of Table 1 by a URI to demonstrate the use of URI

original	<Emma> <is a> <novel>
revision	**<www.dbpedia.org/page/Emma_(novel)>** <is a> <novel>
original	<Emma> <was written by> <Jane Austen>

[12] http://wiki.dbpedia.org/

revision	**<www.dbpedia.org/page/Emma_(novel)>** <was written by> **<www.dbpedia.org/page/Jane_Austen>**
original	<Jane Austen> <is an> <author>
revision	**<www.dbpedia.org/page/Jane_Austen>** <is an> <author>

Knowledge Component. Ontologies can be broken into two parts: the Tbox (terminological component) and Abox (assertion component). The Tbox contains the declaration of classes and properties using vocabularies, which are used to describe knowledge within a domain. Additionally, the Tbox contains rules, constraints and reasoning axioms about those classes and properties. The Abox consists of resources which are instances of the classes defined in the Tbox. The two sub-sections below highlight three important aspects of the T-box that are relevant to this thesis, namely, vocabularies, classes and properties.

Vocabularies. Vocabularies define the concepts and relationships used to describe and represent an area of concern [32]. In practice, RDF is typically used in combination with vocabularies or other conventions that provide semantic information about these resources. To satisfy these different needs, W3C provides various techniques to different forms of vocabularies in a standard format. These include RDF, RDF Schemas (RDFS) [27] and Web Ontology Language (OWL) [33]. For example, `rdf:type` is a construct of RDF which is used as a property to state the relationship between individuals and the class they are linked to. RDFS allows one to define semantic characteristics of the RDF data by providing a semantic extension to the basic RDF vocabulary. The use of ":" and prefixing of `rdf` provides a vocabulary for describing groups of related resources using classes (`rdfs:Class`) and sub-groups within these classes called sub-classes using `rdfs:subClassOf`. It also adds characteristics like domain (`rdfs:domain`) and range (`rdfs:range`) to property resources, which are used to determine characteristics of other resources. Domain and range are discussed in detail in the next section about classes. OWL, further adds to the RDFS and RDF vocabulary. For example, property resources can be identified as `owl:DatatypeProperty`,

`owl:ObjectProperty` or `owl:AnnotationProperty`. More on OWL can be found on its W3C recommendation [34]. OWL rests on the RDFS and RDF vocabularies, hence the usage of `rdf:type, rdfs:range, rdfs:domain` can be made in conjugation.

Table 3: Replacing <is a> predicate by rdf:type in RDF triple to describe the usage of rdf:type

original	<www.dbpedia.org/page/Emma_(novel)> <is a> <novel>
revision	**<www.dbpedia.org/page/Emma_(novel)>** rdf:type <novel>

While RDF, RDFS, and OWL are themselves vocabularies, they also form the basis for creating more vocabularies. Each of these vocabularies can also have variants, called profiles. For example, OWL 2, which is a version of OWL has profiles: EL, QL, and RL. Each of these profiles has a different favourable computational properties. Table 4 has been inspired by Table 1 of W3C RDF Primer 1.1 [29].

Table 4: Table representing common constructs from vocabularies such as RDF, RDFS, and OWL, their syntactic form in an RDF triple and a brief description of the construct

Construct	Syntactic form	Description
rdfs:Class	**C** rdf:type rdfs:Class	**C** (a resource) is an RDF class
owl:DatatypeProperty	**P** rdf:type owl:DatatypeProperty	**P** (a resource) is a Data property
rdf:type	**I** rdf:type **C**	**I** (a resource) is an instance of **C** (a class)
rdfs:subClassOf	**C1** rdfs:subClassOf **C2**	**C1** (a class) is a subclass of **C2** (a class)
rdfs:domain	**P** rdfs:domain **C**	domain of **P** (a property) is **C** (a class)
rdfs:range	**P** rdfs:range **X**	range of **P** (a property) is **X** (a class or datatype for literal)

Classes, sometimes called concepts in an ontology, are a way to represent general qualities and properties of a group of individuals (resources). If a group of objects

has the same traits, that fact should be recognized and a class representing these traits should be created. For example, Novel can be a class with URI <xyz.com#Novel>. Even Novels can have different categories such as Historical Novels, Social Novels. These categories can be modelled as subclass using rdfs:subClassOf construct. Table 5 below extends the example from Table 3.

Table 5: Table shows revision of RDF triple (revision row) followed by addition of RDF triples containing

original	**<www.dbpedia.org/page/Emma_(novel)>** rdf:type <novel>
revision	**<www.dbpedia.org/page/Emma_(novel)>** rdf:type <xyz.com#Novel>
addition	<xyz.com#Novel> rdf:type owl:Class
addition	<xyz.com#SocialNovels> rdf:type owl:Class
addition	<xyz.com#SocialNovels> rdfs:subClassOf <xyz.com#Novels>

Properties, represent relationships (predicate) between subject resources and object resources in ontologies. The rdfs:domain and rdfs:range, constructs of RDFS vocabulary allow to define the scope properties by specifying the domain and range. Domain (rdfs:domain) is an instance of rdf:Property that is used to state that subject resources of a given property is an instance of one or more classes [35]. rdfs:range is an instance of rdf:Property that is used to state that the object resources of a property are instances of one or more classes. The range can either be a class, in which case the property is called an Object Property *(*owl:ObjectProperty*)*. When the Range is a non-URI, literal value e.g. names, date of birth, age; it becomes a Data Property *(*owl:ObjectProperty*)*. This difference in properties is expressed using OWL's owl:DatatypeProperty and owl:ObjectProperty.

Mentioning a data range for data properties enables to restrict the data entered for the concerned data property. For example the range of the property is date. Hence a user cannot enter string or integer type data against the property. OWL uses the RDF data typing scheme, which refers to XML schema datatypes [36].

XML schema datatypes is W3C recommendation that defines datatypes to be used in XML schemas and XML specifications.

Figure 4: shows a snapshot of an ontology where classes such as *Paper* and *Decision* are related to the object property *has decision*. Data property *email* has its domain as class *Person* and range to be `xsd:string`. The ontology begins with optional URI prefixes for common vocabularies. They reduce the effort of re-writing the long URI every time in the rest of the document. *Co-author* is a resource with URI `(<http://cmt#Co-author>)`. Its label is "Co-author" which is identified by `rdfs:label`, another construct of RDFS. *Co-author* class is a sub-class of class *Author*. The concept of sub-classes has been discussed previously in the same sub-section while discussing classes. Figure 3, also describes resource *acceptPaper* (`<http://cmt#acceptPaper>`) with its domain as class *Administrator* and range as class *Paper*. Lastly, Figure 4 describes resource *email* (`<http://cmt#email>`) with its domain as class *Person* and range as a literal with datatype belonging to `xsd:string`.

```
@prefix : <http://cmt#> .
@prefix owl: <http://www.w3.org/2002/07/owl#> .
@prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#> .
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .

###  Class Co-author
:Co-author rdf:type owl:Class ;
           rdfs:label "Co-author" ;
           rdfs:subClassOf :Author .

###  Object Property acceptPaper
:acceptPaper rdf:type owl:ObjectProperty ;
             rdfs:label "accept paper" ;
             rdfs:domain :Administrator ;
             rdfs:range :Paper .
```

```
###  Data Property email
:email rdf:type owl:DatatypeProperty ;
       rdfs:label "email";
       rdfs:domain :Person ;
       rdfs:range xsd:string .
```

Figure 4: a Partial snapshot of an ontology's Tbox indicating the syntax to create classes, object properties, and data properties

The same ontology of Figure 4 can be visualized graphically as nodes and edges using Figure 5.

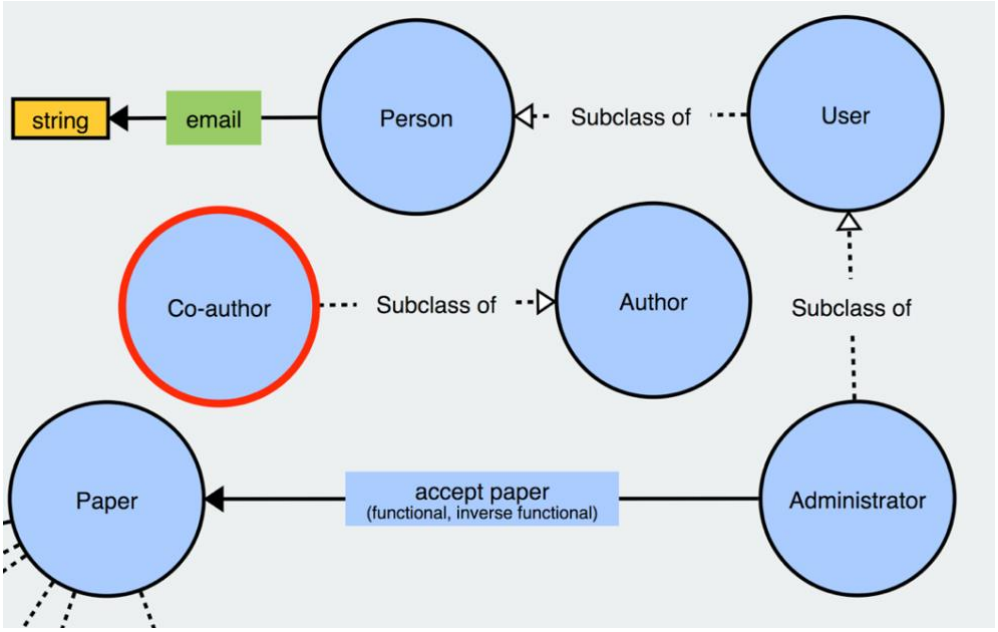


Figure 5: Visualization of the ontology presented in Figure 4

The classes are defined in circles, sub-class property is shown in dotted arrows, the object properties are shown in solid arrows with blue box labels and data properties are shown in solid arrows with green box labels. Figure shows the graphical visualization developed using a web application of Visual Notation for OWL ontologies (VOWL)[13] .

[13] http://vowl.visualdataweb.org/webvowl.html

Section 2.2.1 has discussed the components that comprise an ontology. This includes a description of the RDF data model and the knowledge component of ontologies. The RDF data model introduces the concept of resource, which is identified uniquely by a URI. The data model discusses the RDF triple in which relationships between resources are represented. Further on, the section discussed vocabularies, which provide the concepts that help define relationships between resources. The knowledge component of the ontology was also presented.

2.2.2. Hosting and Querying Semantic Web Ontologies

This section discusses how Semantic Web Ontologies in RDF are stored and queried. This is important as a key aspect of developing a system for relational-to-ontology mapping approach requires retrieval of information from ontologies. Hence the thesis requires the reader to be acquainted with hosting of ontologies on servers and retrieving information using queries. The scope of querying ontologies in this thesis is limited to SPARQL [37] querying.

Triplestores. Linked data is hosted in the form of triples and can be stored either in a flat file or a purpose-built- server called triplestore. Some frequently used triplestores include Apache Jena Fuseki[14], and AllegroGraph [15]. This enables RDF graph data to be easily accessed in a programming environment by the use of SPARQL endpoint [38]. A SPARQL endpoint is a conformant SPARQL protocol service that enables users to query RDF graphs via the SPARQL language.

Query Answering using SPARQL. SPARQL [28] is the query language for RDF graphs. SPARQL is used to express queries in triple patterns. The result of SPARQL queries can be tabular result sets or RDF graphs. A SELECT SPARQL query along with a "WHERE" clause returns variables, which are prefixed using

[14] https://jena.apache.org/documentation/fuseki2/
[15] https://allegrograph.com/

"?" and their bindings directly in a tabular result set. Figure 6 shows a sample SELECT SPARQL query which is modelled on the question - *List all the books with their authors and date of publishing, which have been written in Ireland*. A SPARQL endpoint accepts queries and returns results via HTTP.

```
PREFIX ex: <http://xyz.com/onto#>
SELECT ?book ?author ?date
WHERE
{
?x    ex:authorName       ?author  ;
      ex:isAuthorOf    ?y         .
?y ex:bookName ?book .
?y ex:publishDate ?date .
?book    ex:countryWritten    ex:Ireland  .
}
```

Figure 6: Sample SPARQL query – "List all the books with their authors and date of publishing, which have been written in Ireland"

Section 2.2.2 discussed how ontologies represented in RDF can be stored and accessed on servers. Ontologies are stored on servers called triplestores, which allow programming environments to access data by executing SPARQL queries. The sub-section concluded with an example of a sample SPARQL query.

2.3. Relational Database-to-Ontology Mapping Approaches

Recent years have seen the evolution of the Web as a global information space where linking of information and knowledge from heterogeneous sources and curators is common [25]. Heterogeneity refers to the different structure of data sources e.g. integrating relational database and RDF graph. Relational-to-ontology mapping approach (or uplift) has promised to solve this problem by integrating

different data models under a conceptual knowledge layer in the form of ontology ([39], [40], [41]).

The Relational-to-ontology mapping approach refers to methods and tools that allow relational databases to be represented as an ontology, thereby enabling SPARQL querying or easy integration with other Semantic Web datasets. An ontological data model becomes the target data model, to which the relational database should be mapped to. The ontology may already exist or is modelled keeping the relational data model under consideration. Mappings link the entities and relations of a relational data model, which the source data, to equivalent concepts and properties in the target ontology. The relational-to-ontology approach can be categorized based on the approach to data integration and management. They can either be (1) Materialisation and (2) Virtualisation based [11]. These are explained further below:

Materialization. Transformation of the relational database to ontologies. In this approach, mappings convert the relational database into an ontology which contains both the T-box as well as the A-box. This is useful when completely migrating from a relational database to an ontology. In this case, the relational database becomes redundant and any changes to the data thereafter would be directly made to the ontology. However, if migration is not desired, every time, new rows are added in a table belonging to the database, materialization has to be done again in order to obtain the new ontology with updated individuals (A-box). Standard triplestores can be used in the case of materialization. This is because materialization from the relational database to ontology is a one-time effort. Afterward, the ontology obtained can be accessed as any other semantic web ontology. For example, consider a small size organization (< 500 employees), they have a relational database to store employee details. If they decide to migrate the present database to a triplestore, they will opt for a materialization and deprecate the original database. Any insertion/deletion of data will from then on, will occur in the triplestore.

Virtualization. In this case, the ontology only contains the T-box and the A-box data always remains as rows in relational databases. This is typified by the Ontology-based data access [11] approach. This approach executes SPARQL queries via the conceptual layer of ontology's T-box over the relational database. This involves query re-writing where SPARQL queries are re-written into SQL queries using mappings. The result set from the SQL queries over the relational database is then again transformed to RDF triples using the same mappings. Hence the final output from the SPARQL queries is in the form of RDF triples. This virtual triplestore is not static and can reflect the data changes made in the relational database. In case of virtualization, specialized tools such as Quest Virtual RDF repository[16] using the Sesame Workbench on a Jetty web server bundled with Ontop [11]. More tools include D2R server[17] and Virtuoso RDF views[18]. For example, consider a large company (>500 employees) having a relational database to store employee data, and would like to utilize the prowess of a triplestore. However, several other enterprise systems and software utilize this relational database and it is extremely costly and difficult to migrate each and every software to query a triplestore. In this case, the company will opt for a virtualization where the relational database will continue to exist as a single source of data, while a virtual triplestore will enable selected software/system to leverage the capabilities of the triplestore. Hence any insertion/deletion of data will continue to happen the relational database, query results from the triplestore will simultaneously reflect these updates.

Both the above cases allow enterprises and users to integrate several heterogeneous data sources. They use an ontology to model the domain, hide the structure of the data sources, and enrich incomplete data with background knowledge. The queries (SPARQL) and result set (RDF triples) from querying remain the same in both the cases. The nature of mapping correspondences also remains the same in both

[16] https://github.com/ontop/ontop/wiki/ObdalibQuestIntro
[17] http://d2rq.org/d2r-server
[18] https://virtuoso.openlinksw.com/

materialization and virtualization. Hence the contributions of this thesis can be used in both cases.

2.3.1. Mappings

Mappings ([3], [42], [2], [43], [44]) in the context of relational-to-ontology approaches are declarative statements that specify how entities and relationships of relational databases are encoded in the ontology. The entities and relations in the relational database are the sources of the mapping. The concepts in the ontology become the target for the mappings.

In order to integrate relational databases into ontologies, the W3C RDB2RDF Working Group has ratified two standards to map relational databases to ontologies: Direct Mapping (D2R)[19] and Relational Database to RDF Mapping Language (R2RML)[20] .

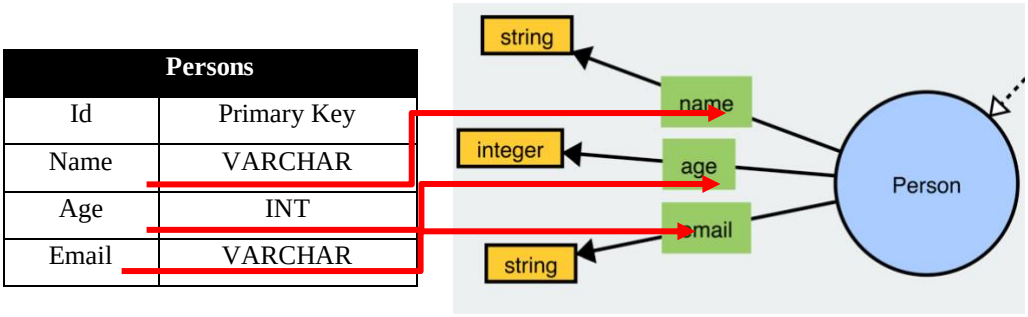
Direct Mappings defines a relational database as an RDF graph. The concept of Direct Mapping is to transform relational tables to classes, foreign key-primary key relationships to object properties and non-key columns to data properties. The scope of Direct Mapping, however, is limited to 1-1 correspondences. In other words it can only match one table to one class, one column to one data property and one FK/PK relation to one object property. The relationship between the relational database (RDB) to RDF has additional complexity since the relational and RDF data models do not exactly align, have different expressivity and each emphasizes a different modelling repertoire [5].

R2RML is a language for expressing customized mappings from relational databases to ontologies. Hence providing the agility to suit the more complex mapping possibilities. For example, one table in the relational database matches to more than one class in ontology. This thesis will not go into the detail of the syntax of R2RML mappings as the contributions made by this thesis are at the mapping

[19] www.w3.org/TR/rdb-direct-mapping
[20] www.w3.org/TR/r2rml

correspondences level, and R2RML is really just one possible concrete expression of such correspondences.

Figure 5 shows an example of relational-to-mapping correspondences in practice. Figure 7(A) shows a part of a relational database schema (an entity or table specification) which acts as the source of the correspondence; Figure 7(B) shows a part of a visualization of an ontology which acts as the target of the mapping correspondence. The mapping correspondence intends to link: (1) the table Persons to the ontology class Person; and (2) columns Name, Age and Email to data properties name, age and email respectively.



(A) (B)

Figure 7: (A) is an example of the source for the mapping correspondence, (B) is an example of the target for the mapping correspondence

Section 2.3.1 discussed mappings in the context of relational-to-ontology mapping approach. It described that the source of mappings are the entities and relationships of relational databases and target of mappings are the concepts of the ontology. It also described the limitations of D2R and how those limitations are overcome by R2RML.

2.4. Chapter Summary

The chapter gave a brief introduction to relational-to-ontology mapping approaches, ontologies, and relational databases.

The key features of the relational database were briefly discussed with particular focus on different classifications of foreign key-primary relationships between tables. These will be important in the thesis as detecting these types of relationships

and relating them to target ontologies is a major goal of the Milan mapping approach. This was followed by an introduction to ontologies, the components such as URIs, classes, and properties that comprise an ontology. The section discussed how ontologies are hosted on triplestores and queried using SPARQL. The last section discussed relational-to-ontology mapping technology and how the approach uses mappings to provide query answering by either materialization of ontological facts or providing a virtual ontology access. This provides the background context to the design and development of Milan's automatic approach to detection relational to ontology mapping correspondences.

3. State-of-the-art

This chapter presents a state-of-the-art review of the research in the field of automatic relational-to-ontology mapping correspondence generation and result of addressing RO1 (Section 1.2). It first discusses the various relational-to-ontology mapping challenges. Requirements are then listed for automatic relational-to-ontology systems to address these challenges. It then describes seven state-of-the-art automatic mapping systems, their capabilities analysed and compared on the basis of the stated requirements. Lastly, gaps in the existing state-of-the-art are identified which motivated the development of Milan.

Section 3.1 presents the relational-to-ontology mapping challenges which are broadly categorized as naming conflicts, structural heterogeneity and semantic heterogeneity. Section 3.2 discusses seven existing state-of-the-art, automatic relational-to-ontology systems. Section 3.3 derives a list of requirements to address automatic relational-to-ontology mapping challenges. Section 3.4 compares the seven existing approaches against the requirements stated in Section 3.3 and identifies the gaps amongst them, thereby laying foundation for development of Milan.

3.1. Relational Database-to-Ontology Mapping Challenges

In the last chapter (Section 2.3.1) the basic concept of relational database to ontology mappings were introduced. In this section the state-of-the-art in mapping approaches are surveyed to identify challenges for identifying and generating mappings. High fidelity relational database (RDB) to RDF mapping correspondences are complex since the relational and RDF data models do not exactly align, the models have different expressivity and each data model emphasizes a different modelling repertoire [5].

A popular set of classifications for mapping challenges in relational-to-ontology patterns, which was first discussed by Batini et al. [24] and subsequently discussed by Pinkel et al [5]. The three classifications are as follows: (1) naming

conflicts, (2) structural heterogeneity and (3) semantic heterogeneity. Each of these three are discussed in the sub-sections below.

3.1.1. Naming Conflicts Challenges

This challenge exists because different data authors and modellers use different naming conventions, even in the same domain. However there are some naming differences that are due to modelling styles that have evolved in the different technology domains of relational databases and ontologies. Names in databases often use short identifiers for tables and attribute names that include technical artefacts for tagging primary-foreign keys while ontologies often use long "speaking names" for their terms [5]. Therefore the main challenge is to be able to correctly match entities with similar names despite the difference in naming convention. Different naming conventions include different tokenization schemes e.g. the use of underscores versus camel case, long naming syntaxes involving repetitive sub-strings across a wide range of related entity labels, and the use of singular vs plural forms. Table 6 lists example of strings that follow different naming convention to represent "*Number of Cars*" as a string label.

Table 6: Different naming conventions to represent "Number of Cars" as a string label

String Label	Type of Naming Convention
`numberOfCars`	`Camel case`
`number_of_cars`	`Delimitter (_)`
`number.of.car`	`Delimitter (.) with singular car`
`Person.number.of.cars`	`Delimitter with extra sub-string`

3.1.2. Structural Heterogeneity Challenges

As discussed in Chapter 1, most real-world relational schema and corresponding ontologies cannot be related using naïve direct mappings as there are vast differences in the ways which the same concepts are modelled. Structural heterogeneity involves the most frequent relational-to-ontology mapping challenges, as it is directly impacted by the way in which data is modelled in either

relational databases and ontologies. Structural heterogeneity can be categorized [5] into three sub-classifications namely: (1) type conflicts, (2) key conflicts and (3) dependency conflicts. The following discusses these three categories of conflicts in more detail.

Type Conflicts

While a normalised relational schema is optimized for storage and data integrity; ontologies are highly abstracted from computational resources and model a domain on the conceptual level [5]. This model diversity causes type conflicts, leading to possibilities of different kinds of structures based on normalization, de-normalization and class hierarchies. The following lists the challenges originating from type conflicts:

1. **Normalization artefacts**, where properties belonging to a single class in the ontology are spread across several different tables in the database.
2. **De-normalization artefacts**, where properties of different classes in the ontology map to attributes in one table in the database.
3. **Class Hierarchies**, while ontologies use hierarchies explicitly, relational databases implement class hierarchies more implicitly. Thus it is a challenge to infer the implicit hierarchy in a database. Hence the mapping follows modelling patterns derived from Chapter 3 of Freitas et al. [45]. An example is shown in Figure 8, which has been adopted from Pinkel et al. [5], uses tables such as *author* and *reviewer* to discuss describe class hierarchies.

In option 1 of Figure 8, a single table corresponds to several sub-classes and the table has additional columns (attributes) to indicate the subclass e.g. column type. According to Pinkel et al. [5], these additional attributes can be numeric type columns for disjoint subclasses or a combination of several type or role flags for overlapping sub-classes. This option is identified by *n:1 class-table mapping challenge* type.

In option 2, there is a table for each specific class to materialize the inherited attributes in each table separately. Both tables have column *name* which matches to

the same data property of class *Person*, thus matching the two tables to a single class *Person.* This option, when multiple tables match to a single class is identified by *1:n class-table mapping challenge* type.

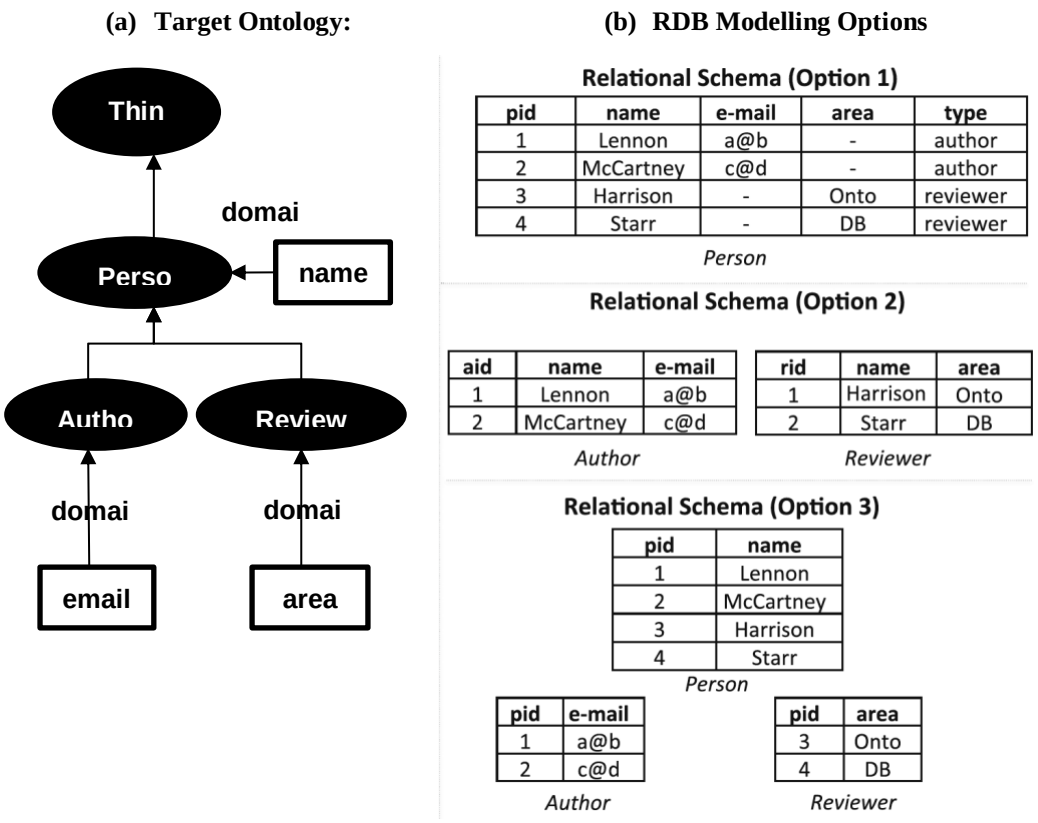


Figure 8: Examples for relational-to-ontology structural mapping challenges. Option 1, is an example of n:1 class-table, option 2 of 1:n class-table and option 3 of 1:1 table-table mapping challenges

In option 3, there is one table for each class in the hierarchy, possibly including abstract super-classes. Tables then use 1:1 table-table, same primary key relationship to indicate the hierarchical relationship. Hence, *pid* connects sub-tables *Author* and *Reviewer* to table *Person*. This relationship helps inherit non-key columns in the super-table to the sub-table.

It is important to note, that the above options depict different data modelling practices that are observed in real-world data modelling and there is no generic favourability associated with anyone these. Each of them have either been adopted

by data modellers as a result of the natural evolution of their databases or it obvious connectivity with other existing processes/systems using the database.

Key Conflicts

In ontologies and the relational data model, keys (record/class identifiers) and references (between classes/tables) are represented differently.

1. **Keys**, in databases are usually implemented using primary keys. Ontologies use URIs as identifiers for individuals. The mapping challenge is to generate URIs from the primary keys when materializing rows of a table into individuals of classes. URIs have global scope whereas keys are local to a database.
2. **References**, in databases is modelled as a foreign key-primary key relationship, ontologies use object properties in OWL or properties that have another class as their range in RDFS.

Dependency Conflicts

These conflicts arise when a group of concepts are related among each other with different dependencies (i.e., 1:1, 1:n, n:m) in the relational schema and in the target ontology. Relational schemas may use foreign keys over attributes as constraints to explicitly model 1:1 and 1:n relationships between different tables. They often model n:m relationships using an additional junction table (Figure 2), which describes a relationship relation. Ontologies may model identifying properties (i.e., functional properties or inverse functional properties) or they define cardinalities explicitly using cardinality restrictions. However, many ontologies do not make use of these restrictions and thus are often underspecified in this respect [6].

3.1.3. Semantic Heterogeneity Challenges

Semantic heterogeneity refers to the contextual or logical differences between conceptual data models of ontologies and relational databases. Three semantic heterogeneity mapping challenges have been defined by Pinkel et al. [5] as the impedance mismatch caused due to object-relational gap. This refers to the difference in modelling techniques where ontologies group information around

objects and properties, e.g. classes where relational database lists them in a series of values structured around tables. It also refers to differences in semantic expressivity. For example databases explicitly represent all concepts, while they can be both explicitly represented and logically inferred in ontologies.

Table 7 in Section 3.3 lists the requirements for each mapping challenge discussed in this section. These are requirements that automatic relational-to-ontology mapping systems should address to generate quality mapping correspondences.

3.2. Existing approaches for Mapping Correspondence Generation

This section describes seven existing approaches from the state-of-the-art literature, compares them with each and finally indicate gaps present in these systems. The seven existing approaches can be broadly divided into Mapping Correspondence Generation Systems (MCGS) and Ontology Agnostic Mapping Systems (OAMS).

MCGS, utilize both the source relational database and the target ontology to establish correspondences. MCGS includes BootOx [9], IncMap [10], MIRROR [19] and Automap4OBDA[15].

OAMS produce putative ontologies from relational databases via Direct Mappings (DM) without taking the target ontology into consideration, then ontology alignment is performed between the putative ontology and the target ontology. OAMS are general purpose systems which include Ontop [11], COMA++ [13] and D2RQ [14].

In order to compare both the type of systems, this thesis adopts the approach used by the RODI relational-to-ontology benchmarking suite [5] by performing this ontology alignment of a putative ontology to the target ontology using the LogMap [46], [47] system for OAMS. This allows OAMS results to be transformed to results from MCGS.

sections below are classified as Mapping Correspondence Generation Systems (MCGS) and Ontology Agnostic Mapping Systems (OAMS), each

containing a description of each of the systems belonging to the specific classification.

3.2.1. Ontology Agnostic Mapping Systems (OAMS)

This section discusses the ontology agnostic mapping systems: Ontop, COMA++ and D2R. They use a source relational database as the input and automatically produce a putative ontology which is agnostic of the target ontology. This means that they can leverage standard ontology alignment tools to map between the putative ontology and the target. In order to evaluate the performance of these systems, LogMap [46], [47] is used to perform ontology alignment between the putative ontology produced by OAMS and the target ontology. The results of these ontology alignments can then be transformed to generate mapping correspondences between the source relational database and target ontology. Subsequent sub-sections discuss LogMap followed by the systems classified as OAMS –COMA++, Ontop and D2R.

LogMap

LogMap is an ontology alignment tool, i.e. it detects correspondences between two ontologies. LogMap is used by systems classified under OAMS to obtain relational-to-ontology mapping correspondences by performing ontology alignment between the putative ontology and target ontology. LogMap uses the Horn propositional logic representation of the hierarchies of each ontology. The Horn propositional representation is used for un satisfiability detection and repair, allowing LogMap to be scalable. It uses the Dowling-Gallier algorithm [48] for satisfiability detection, where the algorithm checks if the source and target resources referenced in a set of mappings, are “satisfiable” with regard to the resources in the source and target datasets. If a referenced resource in an alignment is found to be “unsatisfiable”, then that alignment is classed as invalid. LogMap also attempts to repair each invalid alignment through a repair algorithm. The repair algorithm will search all existing alignments that contain an “unsatisfiable” resource. LogMap uses a reasoning-based approach for the detection and repair of invalid alignments, thereby capable

of handling semantically rich ontologies. It is the choice for ontology alignment in this thesis as it has been used in RODI benchmarking suite as well and the thesis uses the same suite for evaluating Milan. Thus it allows like for like comparisons between all OAMS tools and enables direct reuse of RODI benchmarking results for comparison with Milan's performance.

COMA++

COMA++ [49], which is an extension of the COMA platform [13] is a specialist tool for schema matching. It comprises of multiple matchers that can be combined in a flexible way to automate mapping process. It also allows for users to add new match algorithms to their existing library. It has a "User Feedback Matcher" to enhance the matching results based on user inputs. The match operation takes as input two schemas and determines a mapping indicating which elements of the input schemas logically correspond to each other. Schemas in COMA ++ are represented as rooted directed acyclic graphs and the elements of schema are represented as graph nodes connected by directed links of different types e.g. foreign key-primary key relationship. The authors have performed evaluations using 5 XML schemas, thereby having 10 matching tasks, each matching 2 different schemas. Each of the matching tasks were completed with multiple variants of hybrid matchers.

The evaluation also provides insights into which matchers proves to consistently do better than others. Significant insights from the evaluation results are provided by the author on addressing naming conflicts in automatic relational-to-ontology mapping systems. For example, redundancy in schemas cause instability amongst matchers. This makes the matchers unable to distinguish between different element contexts. Another example is that "TypeName" shows better matching accuracy than "Name". They found that adding the datatype information to the label matching improved the match accuracy. Some of these insights add to the requirements for automatic relational-to-ontology mapping systems. This is discussed later in Section 3.3.

The matchers in COMA++ are however, heavily biased towards addressing the naming conflict challenge amongst the relational-to-ontology challenges. COMA++ only focuses on one-to-one (1:1) match relationships, thereby leaving out all challenges of structural heterogeneity (Table 7).

While mentioning importance of datatype inclusion while matching, COMA++ does not discuss mismatch due to different datatype conventions used in different data models. It also does not provide any particular support for ontologies, thereby making it incapable of addressing challenges pertaining to semantic heterogeneity except in so much as they may be mitigated by the ontology alignment tool used after COMA++.

Ontop

The Ontop Protégé plugin[21] (referred to as Ontop after this section) is a mapping generator developed for the Ontop ontology-based data access framework [11]. It uses direct mapping to obtain a flat putative ontology from the relational database. It is useful for quickly generating mappings, which can be edited by human beings. Ontop's contribution increases when complexity of relational-to-ontology mapping is low, and size of the relational database and ontology is really large. However, it does not perform any lexical, structural or semantic analysis. Hence it falls short in addressing many complex relational-to-ontology mapping challenges. The output is oblivious of the required target ontology, though, so a post-processing with ontology alignment has to be performed for uniform comparison. Ontop is not a tool solely developed for automatic relational-to-ontology mapping. Yet, evaluating with other systems provides an insight into the extent to which complex mappings in scenarios affect the final quality of mapping correspondences.

D2RQ

D2RQ [14] is a mapping generation framework that wraps one or more relational databases into a virtual, read-only graph using a D2RQ's own declarative mapping

[21] https://github.com/ontop/ontop/wiki/ObdalibMappingExtraction

[14]. The `generate-mapping`[22] functionality is responsible for creating D2RQ mapping files by analysing the schema of an existing database. This mapping file is called the default mapping. It maps each table to a new RDFS class that is based on the table's name, and maps each column to a property based on the column's name. This mapping file can be used as-is or can be customized.

D2RQ performs query rewriting from RDF Data Query Language (RDQL) to SQL. The result sets of these SQL queries are transformed into RDF triples. D2RQ does not perform any lexical, structural or semantic analysis as it produces a putative ontology, agnostic of the target ontology. In order to compare the results of D2RQ, LogMap is used as a post-processing step to perform ontology alignment. This feature is still work in progress and comprises of a small part of D2RQ functionality. Similar to Ontop, the purpose of evaluating D2RQ is to gauge the extent to which their lower complexity mapping suffice in real relational-to-ontology mapping scenarios.

3.2.2. Mapping Correspondence Generation Systems (MCGS)

This section discusses the mapping correspondence generation systems: BootOx[9], IncMap [10], MIRROR [19] and Automap4OBDA[15]

BootOx

BootOx [9] is an automatic relational-to-ontology mapping tool that finds one-to-one mapping correspondences for a given source relational database and target ontology. It first creates a putative ontology from the source relational database. Then it enriches this putative ontology, performs ontology alignment using LogMap with the target ontology and finally produces R2RML mappings. Note that LogMap is built-in as part of the BootOx system, unlike the systems classified above under OAMS. Explicit and implicit database constraints from the relational database are encoded as OWL 2 axioms. Table 1 of E. Jimenez-Ruiz et al. [9] shows the encoding of the relational database features supported by BootOx. For example, when foreign key is the primary key while referencing two tables, the corresponding

[22] http://d2rq.org/generate-mapping

classes can be associated with sub-class relationship (`rdfs:subClassOf`). This is followed by enriching the putative ontology with axioms about the classes and properties from these direct mappings.

BootOx uses I-SUB for lexical matching. It has based its choice of lexical matcher on the work of G. Stoilos et al. [50] which has presented comparison of various label matching algorithms in the context of ontology matching. G. Stoilos et al. [50] also highlights good overall performance of the Levenstein's distance as an algorithm for label matching. BootOx also features datatype support to bridge datatypes in SQL and OWL 2. Both the above, equip BootOx to successfully address naming conflict challenges of relational-to-ontology mappings.

Table 1 of E. Jimenez-Ruiz et al. [9] exhibits the capabilities of BootOx to partially address structural heterogeneity mapping challenges. However BootOx is only capable of addressing one-to-one mappings. Hence, for example, it cannot match multiple classes to the same table. E. Jimenez-Ruiz et al. [9] discusses various patterns beyond one-to-one mapping, but BootOx does not address any of those mapping challenges in an automatic fashion. Instead, it offers users to build such complex mapping correspondences.

BootOx works over the maximum variants of ontologies/vocabularies such as different OWL2 profiles, each having or not having a few specialized constructs. This semantic richness allows BootOx to address semantic heterogeneity in automatic relational-to-ontology mapping challenges. The extent however is unquantifiable, due to lack of enough literature stating the ideal performance while addressing the challenge due to semantic heterogeneity.

The authors have evaluated BootOx across the EU Optique project datasets, which includes data from Siemens [51] and Statoil [52] projects. The result of this evaluation shows that BootOx's encoding of relational database features to OWL 2 increases the "query class coverage" results [9] from 27% to 39% in Statoil dataset. BootOx authors have also used the RODI benchmark [5] to compare their results against IncMap, -ontop- and MIRROR.

IncMap

IncMap [10] is primarily a semi-automatic approach for finding one-to-one mappings between relational schemas and ontologies. Without user inputs, it works as an automatic relational-to-ontology tool. Matching in IncMap is based on the Similarity Flooding (SF) algorithm of Melink et al. [53]. A naïve SF algorithm creates a pairwise connectivity graph (PCG) by merging the graphs of the two schemas, representing potential matches. SF is suited where both the schemas follow the same modelling principles. However this is not true in the case of relational databases and OWL ontologies. Thus SF fairs poorly (~20% less) in matching due to this impedance mismatch. IncMap represents both its input, the ontology and relational schema uniformly, using a structure-preserving meta-graph, called IncGraph+.

IncMap [10] is an improvement from it's old version [54], where the old version was not able to find more complex mappings in many real-world scenarios. Incmap [10] leverages knowledge derived from reasoning over the input ontology, thus utilising the semantics of the ontologies. This includes constructs such as `owl:ObjectProperty`, `rdfs:subClassOf`, `rdfs:Domain`, and so on. This equips IncMap to address the relational-to-ontology mapping challenges relating to semantic heterogeneity.

IncMap uses the same relational-to-ontology mapping patterns (Figure 4 of [10]) discussed in the RODI paper [5], which is also used in this thesis. IncMap supports fully automatic relational-to-ontology mapping generation. However IncMap only supports 1:1 and 1:n relational-to-ontology mappings. It does not support any kind of m:n or n:1 relational to ontology mapping correspondences. The IncMap paper does not clarify if the 1:n mappings include both classes and properties or only one of them. The paper does not elaborate on methodologies on how IncMap addresses the mapping challenges posed by the various patterns of structural heterogeneity. Hence the understanding of IncMap's capabilities in addressing structural heterogeneity remains ambiguous (although RODI benchmarking results are available for its performance, see Section 5.2). IncMap in the rest of the thesis from this point will be referred to as IncMap.

Label matching in IncMap ignores hierarchical information, datatypes etc. The matching is performed by the SF algorithm using Jaccard similarity. The paper does not throw very substantive light on how the SF algorithm performs label matching in different relational-to-ontology mapping challenges.

The authors have evaluated IncMap [10] over the RODI benchmark [5] to compare their results against ontop, BootOx, COMA++ and the previous versions of IncMap [54].

MIRROR

MIRROR [19] is a tool for automatic generating of R2RML from a relational database schema. MIRROR has been implemented as a module of the RDB2RDF engine morph-RDB [19]. Their output is produced in two sets. First set produces mapping homomorphic to the ones produced by Direct Mapping (DM). The second set of mapping encodes the implicit information that is contained in relational databases (e.g. subclasses, m:n relationships) but not exposed by directly following the Direct Mapping approach. The system is based on the assumption that the relational database is completely normalized [19]. This is a limitation because many real-world databases are not normalized [5].

MIRROR has been evaluated using Direct Mapping Test Cases[23], which is a test suite provided by the W3C RDB2RDF Working Group. It comprises of a collection of test cases covering various database schemes such as databases without tables, databases with one table, with 1:n relationships, and with m:n relationships. In each of the tests, the triples that can be expected as a result of applying Direct Mapping rules are provided as a gold standard. This only allows the first set of results which are synonymous to the Direct Mappings to be evaluated. However the authors simply observe the presence of second set of results (additional triples) and not evaluate them. This makes the evaluation of MIRROR weak, particularly because Direct Mapping is already an established methodology for relational to ontology mapping generation. The real essence of such automated

[23] www.w3.org/2001/sw/rdb2rdf/test-cases/

systems is to test more complex scenarios, which are not tested and evaluated for MIRROR.

The authors of MIRROR have also listed 9 types of relational patterns between two or more tables. These patterns referred to as catalogue discuss 1:n and m:n table-table relationships using primary-foreign key relations. This allows MIRROR to address class-table correspondences challenges such as sub-class due to 1:1 table-table relationship (option 3, Figure 8). Another example given in the paper is that of a simple primary key-foreign key relationship (2a, Table 1 [19]). It is not very clear how they differentiate this from the first case of 1:1 table-table mapping, unless they assume that an instance of a sub-class may possibly not be an instance of the super-class.

It is interesting to note that the process of mapping correspondence detection is agnostic of the ontology. Hence MIRROR's cannot address naming heterogeneity as it does not perform any lexical matching. Thus it appears that, even if a relational database and ontology were modelled similarly, but had different naming conventions; MIRROR wouldn't be able to detect the correspondences correctly.

While MIRROR does detect some sub-class relationships, structural heterogeneity mapping challenges are not successfully addressed due to the lack of use of data in the relational database and the lack of the target ontology. MIRROR does use constructs from the ontology such as `rdfs:domain`, `rdfs:range` and `rdfs:InverseOf`. While it seems that the usage of these constructs equips MIRROR to address challenges originating due to semantic heterogeneity, MIRROR's ontology agnostic approach inhibits its effectiveness by not considering the features of the ontology while attempting to address the inter-model gap and impedance mismatch. Hence, it is the only system which uses these constructs to encode the relational information without extracting these features from the ontology. For example, on detection of a foreign key-primary key relationship, it will produce an object property and use `rdfs:InverseOf` to represent the inverse of the object property, without considering the object property present in the ontology.

AutoMap4OBDA

AutoMap4OBDA (A4MO) [15] is a system that automatically generates R2RML mappings based on the intensive use of relational source contents and features of the target ontology. A4MO's relational-to-ontology matcher is dependant not only on the database schema, but also on the database content and features of the target ontology. This provides more evidence for matching to take place. A4MO first generates a putative ontology from the source relational database. The algorithms 1 and 2 of A4MO [15] also refer them as graphs. These graphs contain information such as referential relationship, domain and range of object properties, etc. This is followed by ontology alignment between putative and target ontologies using ontology learning and ontology matching. It is interesting to note that the authors' motivation to not use off the shelf ontology alignment tools is that the target ontology, usually specifies domain knowledge on a high-level of abstraction while putative ontologies derived from relational databases describe the syntactical structure on a very low level of granularity. Moreover, different modelling patterns are usually used in ontologies and relational schemas. Therefore, they posit that the use of structural metrics of ontology alignment tools may hinder the detection of correspondences. Hence A4MO uses string based metrics. They change the string matcher for different ontologies based on the work on string similarity metric for ontology alignment by Cheatham et al. [55].

After ontology alignment, A4MO extends the alignments by iterating over the alignment result to find the shortest distance between all pairs of classes using the Dijkstra's algorithm [56]. For example, this enables matching relationships caused by junction tables of relational databases to object properties of ontologies. This approach is similar to the one proposed by Milan, where it used combinatorial optimization to create one-to-one matches from label matching scores (see Section 4.3.2). Ontology learning techniques are applied to infer class hierarchies. The paper mentions that this is done by applying data mining techniques to identify the "categorizing" attributes where sub-classes can be extracted. The two ways in which this is done by A4MO has been mentioned by Cerbah [57] - Identification of

lexical clues in attribute names and filtering through entropy-based estimation of data diversity.

A4MO has been evaluated using the RODI benchmark [5] across 17 scenarios. It has been compared to BootOx, IncMap, ontop, MIRROR, COMA++ and D2RQ. The closest system to A4MO is BootOx as they both have similar transformation rules (based on Direct Mapping), produce putative ontologies and support OWL profiles. The main difference between A4MO and BootOx is that BootOX does not take advantage – in an automatic mode – of the contents of the database to enrich class hierarchy of the putative ontology, instead BootOX has a user-intervention based approach to iteratively improve mappings.

Section 3.2 discussed seven existing approaches which are broadly categorized into Ontology Agnostic Mapping Systems (OAMS) which included Ontop [58], COMA++ [13] and D2RQ [14]; and Mapping Correspondence Generation Systems (MCGS) which included BootOx [9], IncMap [10], MIRROR [19] and Automap4OBDA [15]. Each system was discussed as follows: method and function; evaluation strategy used by the authors of these systems to validate the capabilities of their respective systems; analysis of their capabilities in addressing various relational-to-ontology mapping challenges; and finally the positives and limitations of the system.

3.3. Requirements

This section lists the requirements that have been derived for automatic relational-to-ontology mapping systems through analysis of the challenges discussed in Section 3.1 and the capabilities and limitations of state-of-the-art mapping systems analysed in Section 3.2.

It is relevant to note that the set of challenges discussed above have also been adopted by authors of RODI Benchmark [5] to develop RODI's benchmarking system. RODI contains benchmark scenario (dataset) which contains a source relational database, a target ontology and sets of equivalent SQL and SPARQL test

queries. The SQL-SPARQL query pair is designed keeping the challenges faced by relational-to-ontology mapping systems, to ensure the test can closely mimic the challenges encountered by mapping generations systems in the real world. Since RODI has been used as a benchmark with RODI has been extensively chosen as the evaluation methodology by many state-of-the-art relational-to-ontology mapping systems such as BootOx [9], IncMap [10], MIRROR [46] and Automap4OBDA [15], setting requirements based on the benchmark's challenges will potentially enable design of a robust mapping system.

Table 7 summarizes the requirements of relational-to-ontology mapping approaches for automated systems. Table 7 is adapted from Table 1 of C. Pinkel et al. [5]. The requirements of Table 1 are classified under the same classifications as the challenges mentioned in Section 3.1, namely naming conflicts, structural heterogeneity and semantic heterogeneity.

Table 7: 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 an 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)
ST4a		Indirect k-way n:m relation (class:table)
ST5	**De-normalization**	Correlated entities
ST6		Multi-value
ST7	**Class Hierarchies**	1:n property-column match with type column (option 2, Fig Figure 8)
ST8		n:1 class with type column (option 1, Fig Figure 8)
ST9		n:1 class with type column (option 3, Fig Figure 8)
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 (Figure 2)
SEMANTIC HETEREOGENITY		
SE1	**Object-Relational Gap and Impedance Mismatch**	OWL profile support. E.g. Inverse properties

The requirements R1, NC1 and NC2 are minor additions to Pinkel et al.'s Table 1 [5] to add to the requirements for an automatic relational-to-ontology mapping systems and increase the scope of the naming conflict challenges based on the work of this thesis.

R1 specifies the input and output requirements. The system should take a relational database and an ontology as inputs and produce a mapping correspondence or mapping as an output.

NC1 states that the automated system should be able to detect the naming convention and tokenize the labels suitably for string matching. This requirement is an outcome of a pattern discussed in relational-to-ontology mapping challenges which discussing naming convention. It has also been discussed by several other state-of-the-art systems discussed in Section 3.2 above.

NC2 states that automated system should be capable distinguishing labels based on added information such as datatype while matching data properties to non-key columns of the database. NC1 and NC2 are both types of naming conflict.

ST1 to ST15 define the requirements that automated systems must address in order to overcome structural heterogeneity mapping challenges. These can be classified under normalization, de-normalization, class hierarchies, key conflicts and dependency conflicts. The term `attributes` refers to the relationship between a non-key column of a relational database (RDB) and a data property of an ontology. Similarly, `relation` refers to the relationship between a FK/PK relation and an object property; and `classes` refers to table to class relationships.

Lastly, **SE1** gives the requirement to overcome semantic heterogeneity. Semantic heterogeneity has been discussed in mapping challenges of Section 3.2.

While Table 1 from C. Pinkel et al. [5] from which Table 7 has been adapted, is the most widely accepted compilation of requirements for automatic relation-to-ontology mapping systems, the requirements under structural heterogeneity could have been better explained using supporting examples. Thus Challenge 12 of Table 1 of C. Pinkel et al. [5] has been dropped as it refers to the combination of Challenge 10 and 11.

This section explicitly listed the requirements an automatic relational-to-ontology system should address. These requirements have been adapted from Table 1 of C. Pinkel et al. [5] with the minor additions of requirements R1, NC1 and NC2.

3.4. Gap Analysis: Comparison of Existing Approaches to Requirements

This section compares the state-of-the-art systems presented in Section 3.2 against the requirements stated in Section 3.3. The section begins with a comparison of the seven state-of-the-art systems and their evaluation methodologies. This is followed by discussion of the gaps found in the current state-of-the-art that provided the motivation to develop Milan.

Table 8 represents a tabular comparison of the seven state-of-the-art systems against the requirements listed in Table 7. The double ticks (✓✓) indicate that the requirements are fully addressed while a single tick (✓) indicates partial fulfilment of the requirement. A cross (**x**) indicates the system does not address the particular requirement and a dash (−) indicates there is not enough evidence in the supporting academic papers or system documentation to draw a conclusion.

In addition to summarising the conformance to the requirements, the table also describes basic features about the system such as input, output and evaluation methodology.

Comparison with Milan is added in Table 28, which is a similar structured table as the one below, after appropriately evaluating Milan using the RODI benchmarking system. Four (BootOx, IncMap, COMA++, AutoMap-4OBDA) out of seven systems used both a relational database (RDB) and an

ontology (Ont.) as their input while the remaining three (ontop, MIRROR and D2RQ) only used a RDB.

Table 8: Comparing state-of-the-art systems against requirements

Requirement.	BootOx	IncMap	Ontop	MIRROR	COMA++	D2RQ	AutoMap-4OBDA
Input	RDB, Onto.	RDB, Onto.	RDB	RDB	RDB, Onto.	RDB	RDB Onto.
Output	Map.	Map.	Onto.	Map.	Onto.	Onto.	Map.
Evaluation	RODI	RODI	NONE	DM-T	XML-S	NONE	RODI
NC1	✓✓	✓✓	X	X	✓✓	X	✓✓
NC2	✓✓	✓✓	X	X	✓✓	X	✓✓
ST1	✓	✓	X	✓	✓	X	✓
ST2	✓	-	X	-	-	X	✓✓
ST3	✓	-	X	-	-	X	✓
ST4	X	-	X	X	X	X	✓
ST4a	X	X	X	X	X	X	X
ST5	-	-	-	-	-	-	-
ST6	-	-	-	-	-	-	-
ST7	X	-	X	X	X	X	X
ST8	X	-	X	X	X	X	X
ST9	✓	✓	X	✓	✓	X	✓
ST10	X	X	X	X	X	X	X
ST11	X	X	X	X	X	X	X
ST12	X	X	X	X	X	X	X
ST13	X	X	X	-	-	X	X
ST14	X	X	X	X	X	X	✓
ST15	X	✓	X	X	X	X	✓
SE1	✓	✓	X	✓	X	X	✓

Amongst the four systems mentioned above, AutoMap-4OBDA is the only system that examines the database contents to detect mapping correspondences. All others (BootOx, IncMap, COMA++) only use the schema of the relational database and the ontology (T.box).

The evaluation for each of the systems has also been done differently, with 3 out of 7 seven systems using RODI benchmarking tool for evaluation. MIRROR uses Direct Mapping Tests (DM-T) which only evaluate direct mapping cases. COMA++ uses 5 pair of XML schemas (XML-S). The drawback of this evaluation is the lack of evaluation for specific case of relational-to-ontology mappings. Ontop and D2RQ have not been evaluated for automatic relational-to-ontology mapping.

Amongst all of these, RODI emerges as the most rigorous and comprehensive evaluation methodology. All of the others do not test for many requirements such as inter-model mismatch, structural heterogeneity, impedance mismatch, naming conflicts, and so on, each of which have been included in the RODI benchmarking tool.

Approaches to dealing with naming conflicts have been included well by some authors into their respective systems. BootOx leverages the research on label matching for ontology alignment [50] for its choice of matcher, and emphasizes the good performance of Levenshtein distance [59]. IncMap uses Similarity Flooding algorithm for advanced label matching. AutoMap4OBDA leverages the research on string similarity metrics for ontology alignment [55], which is followed by extending these alignments using ontology learning. All these three systems and COMA++ in some manner stress that unlike ontology alignment, pure label matching will not address naming conflicts as knowledge representation in a relational database is very different from an ontology. Hence each of these take some or all of the additional information such as class hierarchies or datatypes into consideration. This means that systems performing naïve ontology alignment are expected to have poorer results. Ontop, D2RQ and MIRROR do not have its own label matcher.

Most systems are far from fulfilling all requirements for addressing structural heterogeneity. All systems perform up to the level of Direct Mapping, where they can match a table to a class, a FK/PK relation to an object property and column to a data property. All systems except AutoMap4OBDA can detect1-1 correspondences. IncMap claims 1:n relationship, but the paper does not clarify if the 1:n relationship refers to table-class, FK/PK – object property or both. All systems except Ontop and D2RQ support detection of 1-1 table-table relationships, thereby discovering parent table – child table relationships. All systems except Ontop and D2RQ also detect m:n table-table relationship (junction tables), hence allowing matching this relationship to object property of the ontology. Additionally, none of the systems support Indirect k-way n:m relation, where k-junction tables provide class-table associations.

While AutoMap4OBDA claims that it uses entropy-based estimation of data diversity for detecting the 1:n table-class relationship, it is unclear if it refers to the case of column splitting of table by a categorical variable column. Also, none of the systems support m:n table-class relationship. None of the systems except AutoMap4OBDA support detecting the 1:n or n:1 column-data property relationship. All systems have not been designed to function without the foreign key – primary key relationship.

BootOx, IncMap, MIRROR and AutoMap4OBDA have been designed specifically for the relational-to-ontology mapping. Hence this is evident from the support they provide for capturing knowledge and semantics of the ontology. They all provide varying levels of OWL support. This includes constructs such as `rdfs:subClassOf`, `rdfs:range`, `rdfs:domain`. MIRROR and AutoMap4OBDA provide support for `rdfs:InverseOf` as an attempt to address the impedance mismatch caused due to the presence of the inverse of a property while matching it to an object property.

Gaps in State of Art

Despite the fact that some features of the table are unknown, the comparison in this section still clearly shows that there is great scope for improvement. Gaps exist in the current state-of-the-art systems as far as addressing structural heterogeneity is concerned, in particular venturing beyond one-to-one table-class mappings. The research from the current state-of-the-art also shows the importance of detecting 1:1 table-table mappings and junction tables. In addition to the gap in class-table detection using single junction tables, Indirect k-way n:m relation, where multiple junction tables is used is an observed gap in the state-of-the-art. An ideal automatic relational-to-ontology systems should possess a good label matcher which encapsulates the difference in data modelling and naming conventions between relational databases and ontologies. In addition to the string matching in the matcher, basing entity matching decisions on additional information such as class hierarchies, datatypes could improve the performance in addressing naming conflicts. The research also highlights that the usage of OWL schema constructs

increases detection of correspondences. A balanced approach is ideal as the systems should not be biased towards the necessary presence of OWL constructs for matching. Many ontologies used on the web of data might not have all the desired constructs [60]. Lastly, use of the data instances in the database provides rich evidence for detecting mapping correspondences [61] and this approach has only been tried by one state-of-the-art system.

This section compared the capabilities of the state-of-the-art systems against requirements (Section 3.3), with Table 7 providing a high level summary. This included a gap in state-of-the-art analysis. Additionally, certain positive features were described that make individual systems more effective.

3.5. Summary

This chapter introduced the relational database-to-ontology mapping challenges that automatic mapping systems face. The second part of the chapter described the seven state-of-the-art systems that address relational-to-ontology mapping challenges automatically. These systems were picked on the basis of each performing the function of generating mapping correspondences or putative ontologies from relational database. While some involve elaborate lexical, structural and semantic analysis, others focus on simpler mapping challenges. Each of these systems have been evaluated against each other using the RODI benchmarking tool by several authors [9], [10], [15], [5]. This enables this thesis not just to theoretically compare each of these systems but to quantify their performance in comparison against each other and Milan. All systems do not fully support detection of complex correspondences. Also 3 out of 7 systems do not produce mapping correspondences and instead generate an ontology. Third, it lays down requirements of automatic mapping based on the challenges faced by the automatic relational to ontology systems. Fourth it shows that amongst the seven existing approaches, most systems only fulfil a fraction of the mapping

requirements. This shows that there is significant gap for research and scientific work.

This chapter has described the gap which has to be addressed and some of the positive aspects that can potentially make an automatic relational-to-ontology system perform better. Weak coverage of some requirement by SoA systems, together with some positive insights from those systems paved the way for the design and development of Milan's processes and algorithms.

4. Design

This chapter presents the design of Milan, which is a system to automatically generate mapping correspondences between a source relational database and target ontology. It presents the design developed in response to RO2 (Section 1.2).

Milan has been designed keeping in mind the requirements of Section 3.3 (as summarised in Table 7) and the shortfalls identified in Section 3.4 of selected state-of-the art relational-to-ontology systems. Section 4.1 is an introduction to the high-level architecture of Milan, the inputs and the outputs of the system. Section 4.2 and 4.3 then describes the specific and generic sub-processes respectively, that have been used by the main Milan components. Section 4.4 details the three algorithms which constitute the main processes of Milan.

4.1. High Level Architecture

This section introduces the general architecture of Milan, the inputs required by Milan and the output produced by it. Section 4.1.1 discusses the key components and the logical flow of the system. Section 4.1.2 describes the input of Milan. In addition, this section lists all the SQL, SPARQL queries and the meta data retrieved from the input sources in order to analyse them for correspondence detection. Section 4.1.3 describes the template of output of mapping correspondences generated by Milan.

4.1.1. Architecture

Milan has 3 main processes (Table-Class correspondence identification, FK/PK-Object Property correspondence identification and Column-Data Property correspondence identification) that take inputs of a relational database and an ontology (T-box only) to produce a set of mapping correspondences as an output. The relational database constitutes the relational schema, data stored as tuples in the tables in the database. The relational schema comprises of the table meta data such as the column datatypes, primary key(s); and the foreign key-primary key referential relationships. The T-box of the target ontology consists of a document

in RDF, representing the concepts such as classes and properties that are defined in the ontology.

Each of these three main processes generate correspondences by first producing naïve correspondences based on concepts of Direct Mapping (D2R) which maps individual table to class, FK/PK relation to object property and non-key column to data property. This is followed by structural, lexical and semantic analysis to generate complex correspondences, thereby addressing relational-to-ontology mapping challenges. In order to do so, each of these main processes use one or more of six sub-processes that are available to all main processes in the system – label matching, combinatorial optimization, datatype partitioning, junction table detection, property inferencing, column splitting. Table 9 lists the utilization of these sub-processes in the three main processes of Milan. "TRUE" represents that the sub-process is used by the main process while "FALSE" represents that it does not. For example, Junction Table function process is used by the Table-Class correspondence detection process and not by the other two.

Table 9: Utilization of sub-processes by the main processes

Main Processes → **Sub-Processes ↓**	**Table-Class**	**FK/PK-Object Property**	**Column-Data Property**
Label Matching	TRUE	TRUE	TRUE
Datatype Disambiguation	FALSE	FALSE	TRUE
Property Inferencing	FALSE	TRUE	FALSE
Combinatorial Optimization	TRUE	TRUE	TRUE
Junction-Table Detection	TRUE	FALSE	FALSE
Column Splitting	TRUE	FALSE	FALSE

Each of these main processes implement a specific algorithm. Figure 9 shows the process diagram of Milan that comprises of the three main processes. Note that each main process uses the information from the input sources as well as the output correspondences of the previous processes. Finally, the union of Table-Class correspondences, FK/PK-Object Property correspondences and Column-Data

Property correspondences form the full set of mapping correspondences; the output produced by Milan.

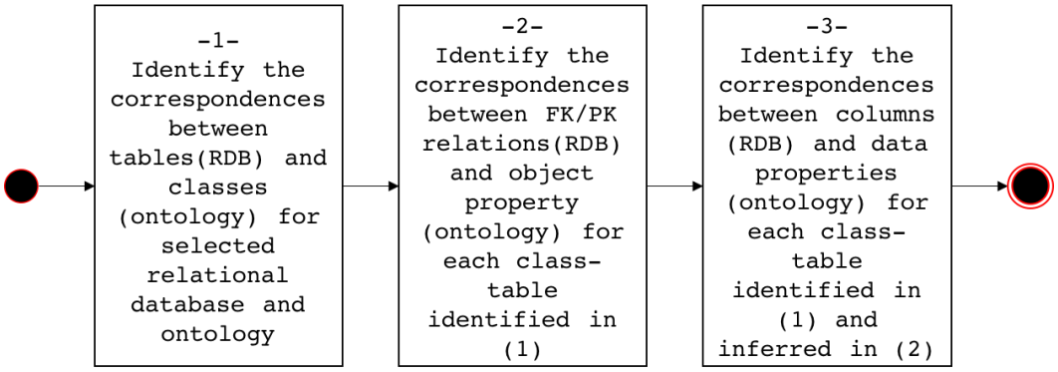


Figure 9: UML process diagram for Milan's main processes

The sequence for the main processes is important as the output of the previous process acts as the input to the next process. Having described the overall architecture of Milan based on three main processes and six sub-processes, the next section describes the input handling.

4.1.2. Input to Milan

The input to Milan is a source relational database and a target ontology. Each sub-process requests different information from the two input sources. Table 10 lists all the inputs extracted from the different parts of the source relational database. The second column indicates the processes and sub-processes that utilize these as inputs.

Table 10: Inputs extracted from various part of relational database along with the processes and sub-processes utilizing these inputs

Input Type	Consuming Sub-Processes/Processes
Table Names	Label Matching, Table-Class Correspondence, Hungarian Algorithm
Column Values	Label Matching, Column Splitting, Hungarian Algorithm
FK/PK Relationships	FK/PK-Object Property Correspondence, Junction Table Detection, Property Inference (1-1 table-table pair), Hungarian Algorithm

Column names and Datatypes	Column-Data property Correspondence, Column Splitting (only column names), Hungarian Algorithm

Similarly, Table 11 lists the consuming processes for the inputs extracted from the input target ontology.

Table 11: Inputs extracted from the target ontology along with the processes and sub-processes utilizing these inputs

Input Type	Consuming Sub-Processes/Processes
Class Labels	Label Matching, Table-Class Correspondence, Hungarian Algorithm
Sub-classes/Sibling-classes	Column Splitting, Label Matching, Hungarian Algorithm
Object Properties and Inverse	FK/PK-Object Property Correspondence, Property Inference (Filter Prioritizing)
Data Properties and Annotation Properties	Column-Data property Correspondence, Hungarian Algorithm, Label Matching

4.1.3. Output of Milan

Milan outputs a set of mapping correspondences between the source relational database and target ontology. The mapping correspondences are generated in a proprietary correspondence format using CSV files, which can be easily translated to R2RML. These correspondences can be classified into three categories. Each of these categories are produced by the three main processes of Milan – table-class correspondences, FK/PK-object property correspondences and column-data property correspondences. Each category of correspondence has a defined tabular data output template (CSV file). These are described below and examples are given in tables Table 12, Table 13 and Table 14. The concepts of the ontology are referred to by the label in these tables, while a map between the labels and URIs is contained in memory by Milan.

Table-class correspondences (Table 12) consist of the three columns. The first contains the source table from the relational database. The second comprises of a

table condition. In case of 1:n or n:1 table-class mappings, this is used to specify the condition of the column and the column value. The third column contains the label of the target class.

Table 12: Milan's output template with example for table-class correspondences

Table	Table Condition	Class
Name of the table which is part of the correspondence	In case of 1:n or n:1 matching, the column and the value in the column becomes the table condition. (Can be NULL)	Label/Fragment ID of the Class which is part of the correspondence
Person	NULL	persons
facility_fixed	fclKind = 'JACKET 12 LEGS'	Jacekt12LegsFacility

FK/PK-object property correspondences (table Table 13) consists of four columns. The first contains the pair of table names connected to each other by primary key and foreign key relationship respectively. The second column contain the join condition, which could be the foreign key column name or junction table name. The third column contains the object property that matches the table pair. The fourth is the type of matching the table pair and object property has. This also helps identify the type of join condition, that is a foreign key column name or junction table name. This is further discussed later in Section 4.4.2.

Table 13: Milan's output template with example for FK/PK-object property correspondences

Table-Table (PK,FK)	Join Con. (JT/FK_col)	Object Properties	Type
Pair of table names connected by primary key – foreign key relationship or a junction table	Name of foreign key column or name of junction table	Label of object property	Type of relationship
administrators, paper	accepted	acceptedBy	D2R

co_authors, papers	co_author_paper	co-writePaper	IJ
co_authors, paper_full_versions	co_author_paper	co-writePaper	i-IJ

Column-data property correspondences (Table 14) consists of four columns. The first column contains the source table name, the second contains the source column name of this table. The third column contains the target data property of the ontology that matches the specified column and the fourth column contains the domain class of this data property. It is important to note that data properties not containing a domain class (using `rdfs:domain`) will never be processed by Milan, hence will never be part of the output correspondences.

Table 14: Milan's output template with example for column-data property correspondences

Table	Column	Data Property	Domain Class
Name of the table which contains the column	Name of the column which is part of the correspondence	Label of the data property which is part of the correspondence	Label of the class which is the domain of the data property
persons	email	email	Person
conferences	name	name	Conference

4.2. Specific Sub-processes in Milan

Each of the main processes in Milan, as can be seen in the architecture, use some sub- processes for common matching tasks. Table 9, while discussing architecture of Milan, describes which sub-process is utilized in a particular main process. 4 out of 6 sub-processes are specific sub-processes namely, Datatype Disambiguation, Property Inferencing, Junction-Table Detection and Column Splitting. They are briefly described here, and in detail while describing the different main-processes in Section 4.4.

Datatype Disambiguation is used in conjunction with label matching in the third main process for detecting columns-data properties correspondences. This sub-process includes the datatype information to reduce disambiguation. This sub-

process has been designed to fulfil the naming conflicts requirement NC2. This is discussed in detail in Section 4.4.3.

Property Inferencing is used in the second main process for detecting FK/PK-object property correspondences. It is used to detect 1:1 table-table correspondences, inverse properties. It also then reasons over the inverse property to correctly match to the FK/PK relation based on the constraints provided in *FilterPriority* of Algorithm 2. This is discussed in detail in Section 4.4.2.

Junction-Table detection is used in the second main process to detect m:n table-table relationships due to junction tables. It then matches the table pair linked via the junction table to the appropriate object property. This is done using the referential constraints of the relational database, stored in the information schema of the database management system. This is discussed in detail in Section 4.4.2.

Column Splitting is used in the first main process to detect table-class correspondences. It detects the 1:n table-class correspondences due to column splitting by identifying columns constituting categorical variables for a specific table. This is followed by label matching and Hungarian algorithm to detect true positives. Label matching is performed using Levenshtein Distance [59] algorithm and Hungarian algorithm, which solves assignment problems, uses the score matrix from label matching to produce 1:1 correspondences by maximizing the sum of individual scores. label matching and Hungarian algorithm are discussed below in Section 4.3.1 and Section 4.3.2. Column Splitting is discussed in detail in Section 4.4.1.

The next section discusses the other 2 sub-processes, which use generic algorithms to support a main process.

4.3. Generic Sub-processes in Milan

2 out of 6 sub-processes, namely, label matching and combinatorial optimization use generic algorithms previously developed and used for varied applications in computer science, economics and mathematics. The sub-sections below discuss each generic sub-process in detail.

4.3.1. Label Matching

Milan uses the FuzzyWuzzy[24] string matching library, which uses Levenshtein Distance [59], producing scores in the range [0,1] to address requirement NC1 of Table 7. Special features such as tokenization for camelCase and special characters such as '_', '#', '*', '.', etc were added to FuzzyWuzzy's label matching[25]. This was done to break complex labels into tokens which are concatenated using a camelCase convention or special characters. This allowed to utilize the power of FuzzyWuzzy's *sort_token_ratio* function which reorders the tokens based on lengths while matching two labels. This feature is particularly useful when different curators while labelling same entities concatenate tokens in different orders. Table 15 below shows some examples of label matching scores achieved by Milan's basic variant of FuzzyWuzzy used in Milan. *FixedFacility* and *facility_fixed* is an example of first tokenizing labels into (*Fixed, Facility*) and (*facility fixed*) . This is followed by using the *sort_token_ratio* to sort the tokens to match the *Facility* labels together and similarly for *fixed.* The Milan FuzzyWuzzy variant is used in detecting 1-1 table-class correspondences and all other FK/PK-object property and column-data property correspondences by comparing labels.

Table 15: Examples of label matching using FuzzyWuzzy with labels and the match score

Label_1	Label_2	Score
FixedFacility	facility_fixed	100
Pipeline	pipLine	93
DevelopmentWellbore	wellbore_development_all	91
FieldArea	fldArea	89
WellboreCasingAndLeakoffTest	wellbore_casing_and_lot	84

A special variant of label matching exists which has some minor changes incorporated from the one described above. This variant is used for column splitting sub-process in algorithm 1. This variant is referred to in this document as *filtered_label_matching*. This variant filters out the superclass label token from its

[24] https://github.com/seatgeek/fuzzywuzzy
[25] https://opengogs.adaptcentre.ie/mathurs/Milan/src/master/Milan%20Code

subclass or sibling label during label matching, thereby improving matching scores. For example, *Oil_pipeline* is a sub-class of *Pipeline*. *Oil_pipeline* matches to table *pipeline* via the column type that has value as '*Oil'.* This trend is also an outcome of extensive analysis done by COMA++ [49]. This has been discussed earlier in Section 3.2. This was evident while working with the filtering strategy. The experience of working with the data shows that *filtered_label_matching* shows better true positive/false negative results while being used in the column splitting sub-process. Table 16 shows that the filtering allows sparser label matching scores therefore improving the possibility of true positives and minimizing false positives.

Table 16: Comparison showing difference in filtered_label_matching variant and FuzzyWuzzy's generic label matching. Example shows gold standard matching to demonstrate the improvement of filtered_label_matching variant of FuzzyWuzzy than the generic variant

Sub-class Labels	Categorical Values	Basic FuzzyWuzzy Score	Sub-class Labels after filtering	After Filter	Gold Standard
OilPipeline	Oil	40.0	Oil	100.0	TRUE
GasPipeline	Gas	40.0	Gas	100.0	TRUE
OilPipeline	Gas	27.0	Oil	0.0	FALSE

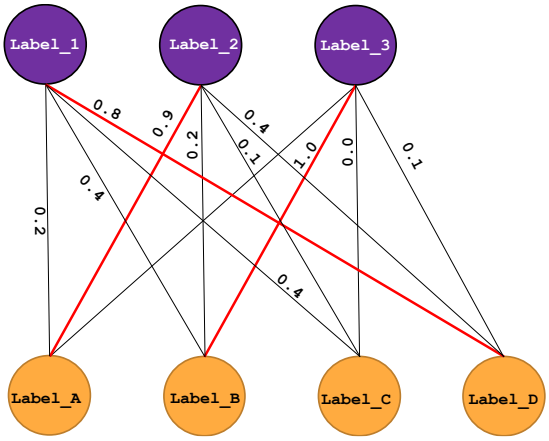
This section has described how label matching is carried out in Milan and detailed two improvements that were implemented to deal with the types of labels encountered in real-world database and ontology deployments, such as the RODI test scenarios [5].

4.3.2. Combinatorial Optimization

Milan uses the Hungarian algorithm [62] to solve the assignment problem across all the three main processes. It consists of finding a maximum score matching

amongst a bipartite graph[26], where edges of this graph represent the label matching scores.

For two sets of labels (x_m, y_n), each having m and n labels respectively, Hungarian algorithm uses the label score matrix of (x_m, y_n) to obtain $min(m, n)$ 1:1 matches such that the sum of scores is the maximum. Figure 10(A) shows two sets {*Label_1*, *Label_2*, *Label_3*} and {*Label_A*, *Label_B*, *Label_C*} represented in a bi-partite graph, the edges across each label indicate the label matching score. The edges depict Hungarian algorithm considers all scores to produce three 1-1 correspondences: (*Label_1*-*Label_D*), (*Label_2*-*Label_A*) and (*Label_3*-*Label_B*). This is because the sum of scores in this case (2.7) is highest for any three scores for non-repeating labels. The red line of Figure 10 shows the 1-1 match as a result of the Hungarian algorithm. Figure 10 (B) the same information as matrix, each cell denoting the label matching score for the label of the given row and column. The red values show the 1-1 match result of the Hungarian algorithm.



(A)

	Label_1	Label_2	Label_3
Label_A	0.2	**0.9**	0.1
Label_B	0.4	0.2	**1.0**
Label_C	0.4	0.1	0.0
Label_D	**0.8**	0.4	0.1

(B)

Figure 10: Example of a use case for Hungarian Algorithm in the context of label matching scores. (A) Representation of the assignment problem as a bi-partite graph, where edges represent label matching score. (B) Representation of (A) as a matrix

Section 4.3 discusses individual use cases of the algorithm in each process. Hungarian algorithm uses the score matrix (representation of a bi-partite graph)

[26] http://mathworld.wolfram.com/BipartiteGraph.html

from label matching sub-process as input to produce 1:1 correspondences by maximizing the sum of individual scores. The score matrix contains label matching score of all combination of matches. This section introduces the fundamental idea of Hungarian algorithm.

Having described the Milan generic sub-processes, we now move on to the main correspondence detection algorithms in the next section.

4.4. Main Process Algorithms

This section describes in detail the algorithms of the three main processes that constitute Milan. Section 4.4.1 describes the detection of table-class correspondences. This includes 1:1, 1:n and n:1 correspondences. Section 4.4.2 describes the detection of FK/PK-object property correspondences. This includes detecting the foreign key-primary key pairs to object properties, detecting junction tables, inferring inverse properties and inheriting object properties by detecting 1:1 table-table relationships. Section 4.4.3 describes the detection of non-key columns and data properties. This includes leveraging SQL-XSD datatype mapping to resolve datatype disambiguation, inheriting data properties using the 1:1 table-table relationship from the previous section.

4.4.1. Table-Class Correspondence Detection

This process takes the class labels, table names and column values to produce the table-class correspondences.

The design of the process is represented using Milan Algorithm 1 and its sub-algorithm, Algorithm 1a. Algorithm 1 first detects 1:1 Table-Class correspondences followed by detection of 1:n table-class relationships using the sub-algorithm Algorithm 1a. These Algorithms are first described in text and using UML Sequence Diagram (Figure 11) and then summarized using an Algorithm listing approach at end of the subsection.

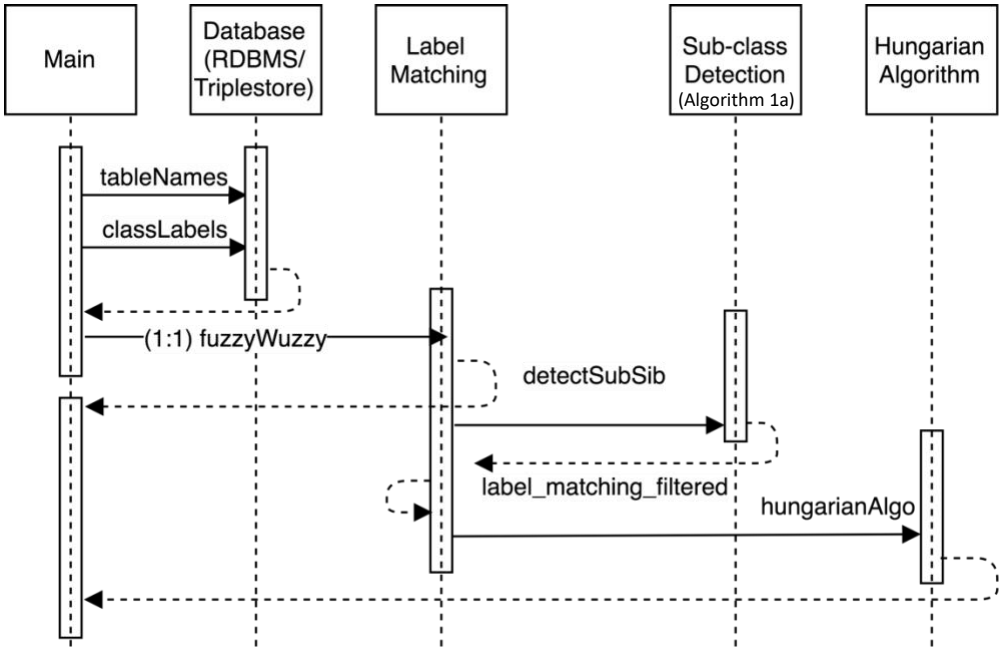


Figure 11: UML sequence diagram depicting the sequence in which sub-processes are executed in Table-Class correspondence detection, which represents both Algorithm 1 & 1a.

Algorithm 1 takes `rdfs:label` of class and name labels of tables as input. It uses the fragment of the class URI in the absence of a `rdfs:label` property value. The Milan label matching sub-process is used over the two lists producing a three-column matching table $O_{a*b,3}$, encapsulating the label matching scores for all combinations of class labels and table names. This is then post-processed by filtering out results with low label match scores based on a threshold, which is determined by trial and error (τ = 0.7). This is followed by filtering out lower match scores due to duplicate occurrences of any class label or table name, represented *as uniqueCriteria* in Algorithm 1. For example, if *label_A* matches *label_B* with score .95 and *label_A* also matches *label_C* with score .75, then the possible correspondence between *label_A* and *label_C* is filtered out of consideration. In case of a class or table having the same score with more than one table or class respectively, but the scores are above the threshold, then all the results are filtered out. This is because there is not enough evidence in such cases to choose one of them. Table 17 is an example of a matching table $O_{a*b,3}$ which is first postprocessed by filtering low score values. The result of this is shown in Table 18 where scores

below 0.7 are removed. Then the step of post-processing removed the lower matching scores that have duplicate occurrences of any class label or table name. This also includes removal of all rows containing *company_lease* class. The results are shown in Table 19 with evidence column indicating the reason for removing some rows. The strike in these tables indicate filtering out of rows in each step. The result of this is a one-one match table. The elimination process continues until a single correspondence is achieved.

Table 17: Table showing an example of the three column matching table O, containing the label matching score between class labels and table names

Class Label	Table Name	Score
BAA_Area	baaArea	1.0
BAA_Area	baaArena	.94
Area_Bay	baaArea	88.0
Company_Lease	companyLost	88.0
Company_Lease	companyLoss	88.0
BAA_Area	barrel	.43

Table 18: Example of postprocessing of Table 4 after filtering out low label match scores

Class Label	Table Name	Score
BAA_Area	baaArea	1.0
BAA_Area	baaArena	.94
Area_Bay	baaArea	88.0
Company_Lease	companyLost	88.0
Company_Lease	companyLoss	88.0
~~BAA_Area~~	~~barrel~~	~~.43~~

Table 19: Example of postprocessing of Table 5 after using uniqueCriteria

Class Label	Table Name	Score	Evidence
BAA_Area	baaArea	1.0	
~~BAA_Area~~	~~baaArena~~	~~.94~~	Repetition of BAA_Area

~~Company_Lease~~	~~companyLost~~	~~88.0~~	More than 1 same match scores for Company_Lease
~~Company_Lease~~	~~companyLoss~~	~~88.0~~	More than 1 same match scores for Company_Lease
~~Area_Bay~~	~~baaArea~~	~~88.0~~	Repetition of baaArea

In an unlikely scenario where two or more correspondences have the same score and cannot be eliminated due to *uniqueCriteria*, all correspondences are eliminated. This scenario is unlikely because, the label matching score of one label with two or more different labels is unrealistic. If this happens, the scores cannot be used to distinguish the probable correspondences.

Unless in a rare scenario where all classes and tables found a 1:1 pair, which is identified by the condition $n < \zeta$, the next step is to address the requirement ST8 of Table 7. Here n is the number of 1:1 pairs found and ζ is the $max(a, b)$ the maximum amongst the number of classes and 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 for the given table-class pair. If there are no sub-classes, then consider the sibling classes ($V_{n,1}$). 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 determined by trial and error, to be $\theta=1.2(n)$, where n is the number of sub-classes or sibling classes. This means that the number of unique elements within the column should be at most, 20% more than the number of sub-classes or sibling classes. The modified variant of Label Matching (*filtered_label_matching*) is then run over this list of unique values with sub-classes or siblings of a target class against all labels of unique column values, using score threshold (τ = 0.7), which has been determined using trial and error. Data diversity is enhanced by 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. As described in Section 4.3.2, Hungarian algorithm uses the concept of assignment problem where maxima of the sum of scores is used to find 1:1 matches between the unique elements of non-key columns and sub/sibling classes.

Table 20, Table 21 and Table 22 illustrate sub/sibling-class detection using example of a source table *Pipline* and target class *Pipeline*. Table 20, lists all the sub-classes and the sibling classes of class *Pipeline.* Count of these sub/sibling classes is denoted by *n*, which is 6 in this case.

Table 20: Table constituting the labels of sub-class and sibling classes for target class "*Pipeline*"

Sub / Sibling Classes	Count (n)
GasPipeline, OilGasPipeline, TransportationPipeline, FeederPipeline, OilPipeline, CondensatePipeline	6

Table 21, comprises of all the columns of table *Pipline*, along with the count of unique values in the column. A simple SQL query using Count(Distinct <column name>) retrieves the unique values. The filtering condition is $\theta=1.2(n)$, which comes out to 7.2. Thus, Table 21 shows unique values of all columns where the number of unique values is less than 7.2. A special variant of label matching followed by Hungarian algorithm is performed between the final sub/sibling class labels and column values.

Table 21: Table showing all columns of table "*Pipline*". Count denotes the number of unique values in the column, <= θ denoted as a Boolean; it depicts if the number of unique values is less than the threshold (θ). If it is "TRUE", the last column lists the unique values for that columns

Columns	Count	<= θ	Unique Values
pipNpdidPipe	59	FALSE	
pipNpdidFromFacility	41	FALSE	
pipNpdidToFacility	41	FALSE	
pipNpdidOperator	6	FALSE	

pipName	58	FALSE	
pipNameFromFacility	41	FALSE	
pipNameToFacility	41	FALSE	
pipNameCurrentOperator	6	TRUE	ConocoPhillips Skandinavia AS, GASSCO AS, GASSCO AS, Statoil Petroleum AS, A/S Norske Shell, GDF SUEZ E&P Norge AS
pipCurrentPhase	3	TRUE	IN SERVICE, DECOMMISSIONED, INSTALLATION
pipMedium	4	TRUE	Oil, Gas, Condensate, Oil/gas
pipMainGrouping	2	TRUE	Transportation, Feeder
pipDimension	18	FALSE	

Table 22 shows the final results after the Hungarian algorithm. The final correspondences are then derived from this, matching the sub/sibling class to a part of the table, which is specified by a condition for the given column having the matching column value. Table 12 shows the template for expressing this as class-table correspondences.

Table 22: Table showing results post Hungarian algorithm and *filtered_label_matching* across list of sub/sibling class labels and the column values labels belonging to different column

Sub/Sib Classes	Column	Column Values	Score
GasPipeline	pipMedium	Gas	100.0
OilGasPipeline	pipMedium	Oil/gas’	92.0
TransportationPipeline	pipMainGrouping	Transportation	100.0
FeederPipeline	pipMainGrouping	Feeder	100.0
OilPipeline	pipMedium	Oil	100.0
CondensatePipeline	pipMedium	Condensate	100.0

Algorithm 1 and 1a summarize the process algorithmically. Line 8 of aalgorithm 1 uses algorithm 1a (subClassDetection) as a function.

Algorithm 1 : Detecting Class – Table Relationship

Procedure: **getClassTableMain** *(X,Y)*
$X_{a,1} \leftarrow$ List of all Classes
$Y_{b,1} \leftarrow$ List of all Tables
$O_{a*b,3} \leftarrow$ **labelMatching(***X,Y***)**
$O_{n,3} \leftarrow O_{a*b,3}$ *where* $O_{a*b,3}[,3] > \tau$ & **uniqueCriteria**
if*(*$n < \zeta$*)*
    For each *i* in *n*
        $M_{a,3} \leftarrow$ **subClassDetection(**$O_{n,3}[i,1]$, $O_{n,3}[i,2]$**)**
        $N_{*+a,3} \leftarrow$ **V.concat(**$N_{*,3}$, $M_{a,3}$**)**
$N_{p,3} \leftarrow$ **V.concat(**$N_{p,3}, O_{n,3}$**)**
Return $N_{p,3}$

Algorithm 1a : Detecting 1:n Table-Class Relationships

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

This section discussed table-class correspondence detection. It used algorithm listing and UML sequence diagram to demonstrate functioning and sequence of execution of sub processes such as label matching, sub-class detection and Hungarian algorithm.

4.4.2. FK/PK-Object Property Correspondence Detection

This process takes a pair of tables linked by foreign key-primary key (FK/PK) relationship, the target ontology and any class-table correspondences identified by algorithms 1 and 1a as input, in order to identify correspondences between FK/PK relations and object properties. The target ontology constructs considered are object

properties, their inverse properties and the range and domain classes of the object property. This process implements Algorithm 2. The algorithm is first described in text and using a UML Sequence Diagram (Figure 12) and then summarized using an Algorithm listing approach at end of the subsection.

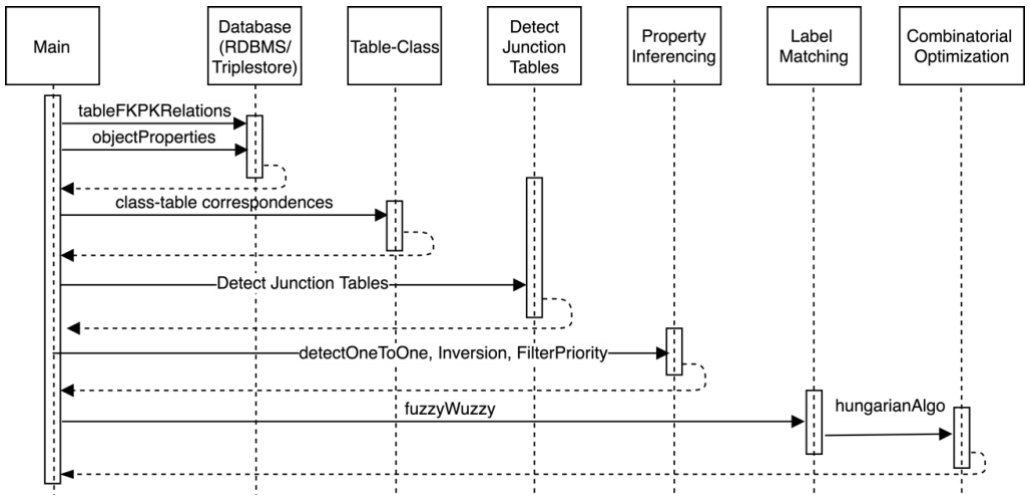


Figure 12: UML diagram depicting the sequence of execution of sub-processes for FK/PK-object property correspondence detection

The algorithm first infers 1:1 table-table relationships in order to allow inheritance of FK/PK relations to sub-tables. This is followed by detection of m:n table-table relationships using RDB junction tables. These two steps that generalize the table-table relationships enables two classes which are not directly linked using FK/PK relationship to be paired. This is necessary in order to detect a potential corresponding object property linking the classes in the ontology. In the case that there are multiple FK/PK relationships or multiple object properties between two classes, label matching followed by the Hungarian algorithm are used to identify the most appropriate correspondence. Figure 12 is a UML sequence diagram that shows the sequence of execution of sub-processes involved in this process.

Algorithm 2 uses the set of FK/PK relationships (R) from the relational database as one of the inputs. The second input is from the ontology where the algorithm retrieves the object properties and their inverses (`owl:inverseOf`) along with the domain (`rdfs:domain`) and range (`rdfs:range`). This is

encapsulated in variable O. Using the table-class correspondences from algorithm 1as the third input, algorithm 2 detects the corresponding two pairs of classes and tables. In the case of two or more FK/PK relations or object properties, algorithm 2 matches the obtained list of object properties with keys of the relational database using label matching followed by Hungarian algorithm. Table 23 shows the three inputs. Using table-class correspondences (1), (3) can be verified to be the expected match if (2) exists.

Table 23: Inputs to FK/PK-object property detection

1	Table	Class		
	person[2]	Persons[2#]		
	Department[1]	Department[1#]		
	paper	Paper		
2	Table- Primary Key	Table- Foreign Key	Foreign Key Column	
	Department[1]	Person[2]	p_department	
	paper	person	author_of	
3	domain	Range	Object Property	InverseOf
	Department[1#]	Persons[2#]	p_department	
	Persons	Paper	authoredBy	authorOf

The correspondence detection described up to here is naïve, as it will miss out on detecting complex relationships such as junction tables, one-to-one table relations, and inverse relationships. Therefore, after the naïve detection of correspondences, Algorithm 2 detects junction tables (Figure 8) by detecting tables where two foreign keys of the table form it's composite primary key. Junction table sub-process scans through the information schema of tables to find tables where there are only two columns, both of which are primary keys and foreign keys. This enables the possibility of many-to-many relationship between two tables. It is also possible that and infers a many-to-many relationship across the table pair. The label used to describe this relationship is the junction table's name.

The property inference sub-process of Algorithm 2 refers to option 3 of Figure 8, where two tables are linked to each other such that the primary key and the

foreign key is the same for the sub-table. The definition of sub-table was provided in Section 3.1.2. The property inference sub-process detects 1:1 table-table mappings and inherits all columns of parent table to the child tables. An example of this pattern is demonstrated in Table 24, which lists the foreign key – primary key relationship across two pair of tables with the primary key and foreign key columns. The last column indicates the primary key of the foreign key table. On inspecting the first two rows of Table 24, it is seen that the foreign key column and the primary key column are the same for the foreign key tables *Journal_Papers* and *Conference_Paper*. Hence a 1-1 table relationships can be inferred amongst table pairs; (*Document-Jorunal_Paper*) and (*Document-Conference_Paper*). Thus, it is said that *Journal_Papers* and *Conference_Paper* are sub-tables of *Document*.

Table 24: Primary key and foreign key relationship across two tables. Table shows the FK/PK respective table and columns. Last column (pk_fktable) lists the primary key of the table containing the foreign key in the FK/PK relationship.

pktable	pkcolumn	fktable	fkcolumn	pk_fktable
Document	Id	Journal_Paper	Id	Id
Document	Id	Conference_Paper	Id	Id
Author	Id	Document	Author	Id
Reviewer	Id	Document	Reviewer	Id

All relationships of Table 24 are represented as an ER diagram in Figure 13.

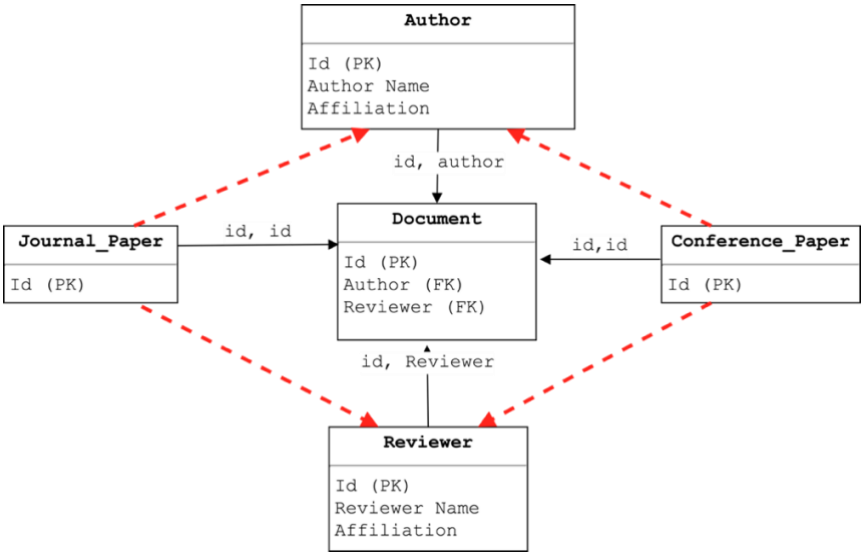


Figure 13: Entity Relationship (ER) diagram representing the foreign keys, primary keys and referential relationships of tables depicted in Table 24

Document has two foreign keys, linking column *Author* to primary key Id of table *Author*. Similarly, column *Reviewer* is linked to primary key Id of table *Reviewer*. These FK/PK relationships of *Document* are inherited by the sub-tables, *Journal_Papers* and *Conference_Paper*. This allows table (*Author, Journal_Papers*) and (*Author, Conference_Papers*) to be matched to object properties that exist between their corresponding classes (result of Algorithm 1).

Table 25 below lists the FK/PK relations inherited by sub-tables by virtue of 1-1table-table relationship. In such cases, table pair *Author-Journal_Paper* relationship can match to the corresponding class pair.

Table 25: Relationships inherited due to 1-1 table-table mapping

pktable_name	pkcolumn_name	fktable_name	fkcolumn_name	Property Inference
Author	id	Journal_Paper	Document .Author	1-1 table
Reviewer	id	Conference_Paper	Document .Reviewer	1-1 table

Lastly, while relationships in relational databases have a clear sense of directionality, relationship across the classes via object properties may not always be uni-directional. Directionality in this context refers to primary key to foreign key relation in relational database or domain to range in ontology. Ontologies may

contain the property and its inverse both (using `owl:inverseOf` construct of OWL). Table 26 outlines the various possibilities where the FK/PK relation may not directly match to the object properties. These possibilities refer to requirement SE1, caused due to impedance mismatch. Milan overcomes this impedance mismatch by dropping directionality (Algorithm 2, line 6) and prioritizing direction (Algorithm 2, line 10) in the presence of alternates.

Table 26 demonstrates the different cases Milan infers from object property and its inverse property In Table 26, class C_1 is linked to class C_2 via object property A, with C_1 as domain and C_2 as range. The table pair corresponding to C_1, C_2 from algorithm 1 is T_1, T_2 respectively. Row (1) shows that the direction of the property and FK/PK relation is $C_1/T_1 \rightarrow C_2/T_2$. Row (2) presents that the relationship in the opposite direction. In (a) of Table 26, column OP lists object properties A and X where X is the inverse property of A. However, the FK/PK relation represented as a in column Col, exists only in one direction (1). This matches A and a, but X does not get matched to any FK/PK relation. Milan's property inference sub-process infers a to X (shown in red). In case(b), 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 special case of junction table relationships, where more than one relation exists across two tables. It could occur that both the object property A and 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.

Table 26: Three cases of inferring FK/PK relation from object property and its inverse property

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)

It is indeed possible to have multiple FK/PK relations or object properties for a single pair of tables or classes respectively. Additionally, the FK/PK relations are non-homogenously present in different directions. Inferences are made to include

missing FK/PK relations for each relation (case c of Table 26). The final matching is performed by label matching and then performing Hungarian algorithm with an additional optimization constraint for each property and its inverse pair :

$$labelMatch(Property, (FK/PK)_a) \gg labelMatch(inv: Property, (FK/PK)_b)$$

This constraint is in reference to case c of Table 26. It means that the label matching score between an object property and FK/PK relation `A` will always have precedence over the label matching score of the inverse of the same object property and FK/PK relation `B`. This is to prevent a property and its inverse, both to occur as a result of the Hungarian algorithm.

Algorithm 2 below summarizes the process.

	Algorithm 2 : FK/PK - Object Property Correspondence Detection
1:	Procedure: **opRefInt**(R, O, N)
2:	$R \leftarrow$ FK/PK relations of RDB
3:	$O \leftarrow$ Object Property and rdfs:InverseOf relations
4:	$N \leftarrow$ Class-Table relationship from Algorithm 1
5:	$R \leftarrow R \cup$ **DetectJunctionTables(**R**)**
6:	$R \leftarrow R \cup$ **PropertyInference.DetectOneToOne(**R**)**
7:	$R \leftarrow R \cup$ **PropertyInference.Inversion** (R) with inversion annotation
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:	$P, C \leftarrow$ **PropertyInference .FilterPriority**(P, C)
11:	$M_{\mathrm{n},3} \leftarrow$ **fuzzyWuzzy**(P, C)
12:	$N \leftarrow$ **HungarianAlgorithm_constraint(**$M_{\mathrm{n},3}$**)**
13:	Return N

This section discussed the FK/PK-object property correspondence detection main process. It described how the process takes a pair of tables linked by foreign key-primary key (FK/PK) relationship, the target ontology and class-table correspondences identified by table-class correspondence detection process as input to identify correspondences between FK/PK relations and object properties. This is supporting using a UML process diagram and algorithm listing. In addition, it discussed various capabilities of Milan to address challenges 1-1 table-table relationship, inverse property inferencing

4.4.3. Column-Data Property Correspondence Detection

This process produces non-key column-data property correspondences. The inputs to this process come from the table-class correspondences from Algorithm 1, property inferences from Algorithm 2 and column meta-data, data properties, property domains and ranges from the input RDB and ontology.

The process first retrieves the table-class correspondences. It also matches the sub-tables which are involved in 1-1 table-table relationship to classes. This is done

by inheriting the columns of the main table to the sub-table. These relationships are retrieved from the 1-1 table-table relationship result of Algorithm 2. Each table-class pair then becomes the basis for matching the columns of the table or the inherited columns to the data properties involving that particular class in its domain. Matching the columns to the data properties then happens in groups divided by datatype. For example, matching columns containing date to the object properties whose range (`rdfs:range`) is also date. Since the convention of predefined datatypes in relational databases and RDF are different, this process leverages the mapping between SQL datatypes and RDF datatypes provided by ISO/IEC 9075-14:2008 Part 14 XML-Related Specifications [63]. Table 27 is borrowed from the table in [63].

Table 27: Mapping predefined SQL Datatypes to XML Schema Types

SQL datatype	RDF datatype
NUMERIC, DECIMAL	xsd:decimal
SMALLINT, INTEGER, BIGINT	xsd:integer
FLOAT, REAL, DOUBLE PRECISION	xsd:double
BOOLEAN	xsd:boolean
DATE	xsd:date
TIME	xsd:time

For each of the datatype groups, a label matching is performed using FuzzyWuzzy. This is followed by implementation of the Hungarian algorithm over the label matching score results to produce a 1-1 column-data property correspondence for each datatype category. The process is defined as a UML sequence diagram showing the sequence of execution of various sub-processes for producing the FK/PK-object property correspondences.

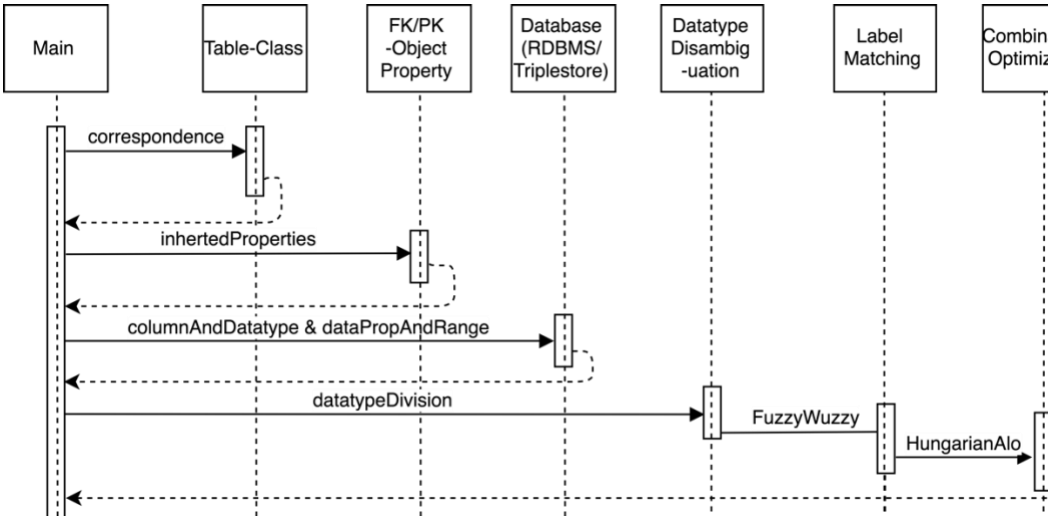


Figure 14: UML diagram depicting the sequence of execution of sub-processes for FK/PK-object property correspondence detection

Algorithm 3 below provides an algorithmic listing. The key approach of this algorithm is to process the data and annotation property along with its domain; and range and non-key columns and their datatypes for a given pair of class and table identified by Algorithm 1. It matches these data properties to non-key columns using label matching and Hungarian algorithm to produce 1-1 correspondences.

Algorithm 3 : Columns and Datatype Property Correspondence

Procedure: **DpC**(X,Y);
$X_{n,1} \leftarrow$ List all classes
$Y_{m,1} \leftarrow$ List all tables
For each i in X
    For each j in Y
        $D_{a,2} \leftarrow$ **Datatype_Annotation Property_&Range**($X_{n,1}[i]$)
        $C_{b,2} \leftarrow$ **Non_Key_Columns_&Datatype**($Y_{m,1}[j]$)
        For each k in ($a \cup b$) using RDB – RDF datatype mapping
            $M_{\mathrm{n},3} \leftarrow$ **fuzzyWuzzy**($D[,1], C[,1]$)
            where $D[,2] = C[,2] = k$
            $N \leftarrow$ **HungarianAlgorithm**($M_{\mathrm{n},3}$)
Return **All** N

This section discussed column-data property correspondence detection. It used algorithm and UML sequence diagram to describe the sequence of execution of various sub-processes to detect column-data property correspondences.

4.5. Implementation and Deployment

The algorithms of Milan have been implemented in the Java programming language and is hosted on ADAPT Centre's OpenGogs repository[27]. OpenGogs is a hosting service for source code management and distributed version control. The java code accesses the relational database using JDBC drivers[28]. The implementation has support to access both PostgreSQL[29] as well as MySQL[30]. The code accesses the ontology using the Apache Fuseki triplestore[31] via its RESTful (REST) API. The results are currently produced by each algorithm as CSV files. The output of some algorithm becomes input or part of the input for the other algorithms. This was been discussed in detail in Section 4.4.

Milan has several applications in diverse fields of engineering, science and management. It is an automatic relational-to-ontology mapping system. Its application is in integrating heterogeneous databases and enabling interoperability, with the use of ontologies and mappings. It's automatic mapping generation feature makes integration and interoperability tasks of modern digital enterprise faster, reducing human effort and dealing with ever evolving data both in size and structure. The application of the above tasks applies to multiple domains including computer science, business, economics and mathematics.

4.6. Reflection

The development of Milan was an iterative process. Challenges were faced to implement each of the algorithms. Algorithm 1 involved challenges to detect n:1 class-table correspondences using column splitting. Identification of column values

[27] https://opengogs.adaptcentre.ie/mathurs/Milan/src/master/Milan%20Code
[28] https://dev.mysql.com/downloads/connector/j/
[29] https://www.postgresql.org/
[30] https://www.mysql.com/
[31] https://jena.apache.org/documentation/fuseki2/

that can possibly match sub-classes required an optimal filtering condition to increase the matching accuracy using label matching.

Algorithm 2 involved most implementation challenges. Detecting junction tables from the relational database required to analyse the FK/PK meta data from the relational database. 1:1 table-table relationships required the same in addition to implementing the object property inheritance. Another challenging part of this algorithm was the property inference by taking into consideration the impedance mismatch between object properties and FK/PK relations. FK/PK relations are non-homogenously present in different directions. Inferences were made to include missing FK/PK relations for each relation (case c of Table 26). The final matching was performed by label matching followed by Hungarian algorithm with an additional optimization constraint to rank label matching scores between an object property and FK/PK relation higher than the inverse of the same property. This is to prevent a property and its inverse, both to occur as a result of the Hungarian algorithm. It was challenging to theoretically build an additional optimization constraint within the combinatorial optimization (Hungarian algorithm) and reflecting the same in the code in Java.

The label matching process in class-table correspondence detection currently extracts the label from either the URI of the classes or `rdfs:label` to then uniquely identify the class in the rest the code. This can cause problems when two or more classes having different URI, have the same label. This is currently a known limitation of Milan. It assumes that there is a unique combination of label and base URI in the ontology. A theoretical solution to address this challenge is to create maps[32] is to link the labels to URI. This will allow classes to be identified by URI, making their identification unique and independent of label. However, this has not been implemented in this thesis, and hence remains as future work.

[32] https://docs.oracle.com/javase/8/docs/api/java/util/Map.html

4.7. Milan with respect to level of support for mapping challenges

Table 28 depicts the theoretical extent to which the state-of-the-art systems are designed to address the mapping challenges for automatic mapping generation systems outlined in Table 7.

Milan utilizes both the source relational database and target ontology. It also has a dedicated label matcher (4.3.1) which has customized variants based on specific relational-to-ontology patterns. Table 28 provides positive indications as to how Milan's capabilities can address structural heterogeneity, as it promises to address normalization challenges, class hierarchies and dependency conflicts. This is a significant design enhancement from the state-of-the-art systems. It is also designed to address semantic heterogeneity, arising due to impedance mismatch. Milan's support for OWL constructs also promises better performance at bridging the inter-model gap.

Table 28: Milan's support to address requirements for automatic relational-to-ontology mapping correspondence generation

R#	Challenge Type	RDB Pattern	Level of Milan Support
R1	Input-Output	System should accept relational database and ontology as inputs and return mappings/mapping correspondences as output	Completely
NAMING CONFLICTS			
NC1	Tokenization	Matching label via different tokenization conventions	Completely
NC2	Datatype Disambiguation	Distinguishing datatype property – column labels based on datatypes. (e.g. xsd:date cannot match to an integer label)	Completely
STRUCTURAL HETEROGENITY			
ST1	Normalization	Weak entity table	Completely
ST2		1:n attribute (class:table)	Partially
ST3		1:n relation (class:table)	Completely
ST4		n:m relation (class:table)	Partially
ST4a		Indirect k-way n:m relation (class:table)	Not
ST5		Correlated entities	Not

ST6	De-normalization	Multi-value	Not
ST7	Class Hierarchies	1:n property-column match with type column (option 2, Fig Figure 8)	Completely
ST8		n:1 class with type column (option 1, Fig Figure 8)	Partially
ST9		n:1 class with type column (option 3, Fig Figure 8)	Partially
ST10	Key Conflicts	Plain composite key	Not
ST11		Missing keys	Not
ST12		Missing reference	Not
ST13	Dependency Conflicts	1:n attribute	Partially
ST14		1:n relation	Partially
ST15		n:m relation represented by junction table (Figure 2)	Completely
SEMANTIC HETEREOGENITY			
SE1	Impedance Mismatch	OWL profile support. E.g. Inverse properties	Partially

These comparisons and claims of course are purely from a design perspective. It is imperative to employ an experimental evaluation that uniformly evaluates each system against the requirements. The next chapter describes an experimental evaluation using a benchmarking tool and benchmarking dataset to compare the quality of automatic relational-to-ontology mapping correspondence generation of Milan with other state-of-the-art systems.

4.8. Summary

This chapter discussed the design of Milan, which is a system to automatically generate mapping correspondences between a source relational database and target ontology. The chapter discussed the general architecture of Milan before describing in detail each of its 3 main processes and 6 sub-processes. Each of the main process utilizes the output generated by other main processes, partially or completely to produce their respective output. The chapter uses examples, illustrations, UML diagrams and algorithm listings to describe in detail the functioning of Milan. Finally it presents an argument as to how Milan addresses the requirements of Table 7 (Section 3.3) and fills the gaps present in the state-of-the-art.

5. Evaluation

This chapter describes an experimental evaluation of Milan. The purpose of this evaluation is to test the quality of mapping correspondences generated by Milan across datasets that can closely mimic the challenges encountered by mapping generation in the real world. This is addressing RO3, which was presented in Section 1.2.

Relational-to-ontology mappings are typically not evaluated using a common benchmark. Instead, authors end up using scenarios and datasets of their own choice. TPC benchmark [17] filled this gap, however no results were reported so far by them. Tarasowa et al. [18] proposed a similar, generic approach to quality measurement on relational-to-ontology mappings. However, amongst these, RODI offers comprehensive benchmarking, with real world datasets, single-globally comparable scoring measure for mapping quality. Additionally, the state-of-the art mapping systems that compare against Milan have been evaluated over RODI. Hence RODI has been used as the benchmarking tool for this thesis.

Mapping quality, as defined by the RODI benchmarking approach, is reflected by an observed score which is the mean of per-test F-measure scores, that is it is a measure of the mappings ability to correctly support information retrieval (IR) tasks in the mapped domain compared to the same IR tasks in the original domain.

The evaluation of Milan is in the form of a lab-based experiment based on the RODI benchmarking methodology. Four RODI input scenarios were selected to offer realistic and varied mapping challenges which holistically test the Milan mapping patterns such as 1:n class-table relationship, 1:1 table-table matching, and so on as discussed in Chapter 3. Each input benchmark scenario (dataset), provided by RODI, contains a source relational database, a target ontology and sets of equivalent SQL and SPARQL test queries. The mapping quality of Milan was evaluated and compared with seven other state-of-the-art automatic mapping generation systems using the observed RODI mapping quality scores. The results of the other seven systems are based off published results. The result of the

comparison indicates that Milan outperforms the seven other state-of-the-art systems for which comparable RODI benchmarking data is available.

The structure of the rest of this chapter is as follows: Section 5.1 presents the Experiment undertaken on the Milan implementation by first describing the Motivation, Hypothesis, Methodology, Datasets and Experimental Setup. Section 5.2 details the results of the observed performance of Milan against the state-of-the-art systems. Section 5.3 analyses the experimental results and finally Section 5.4 presents the conclusions and summarizes the chapter.

5.1. Experiment: Milan Mapping Quality Evaluation

This section describes the experiment conducted to produce results for the quality of mapping generated by Milan over benchmark datasets, which are referred to as scenarios. Section 5.1.1 starts with a hypothesis for mapping quality. This is followed by a description of the experimental setup discussion of approach and scope, in Section 5.1.2. The description of the four scenarios is provided in Section 5.1.3 along with the different challenges contained in them. Section 5.1.4 mathematically describes the quality metrics used to represent the experimental results which are based on the challenges discussed in Section 5.1.3. Lastly, Section 5.1.5 describes in detail, the steps of the experiment to finally compute the quality metrics for Milan.

5.1.1. Hypothesis

Milan generates higher quality mapping correspondences than state-of-the-art systems across four selected scenarios

5.1.2. Experiment Scope

The central framework used for the evaluation of this work is the Java-based RODI benchmark tool. RODI is a benchmarking tool used to evaluate the quality of mapping correspondences produced by automatic relational-to-ontology systems. This benchmarking tool has been used in this research to evaluate mapping correspondences generated by Milan and then compare the results against those of seven other state-of-the-art systems.

The RODI benchmark comprises of (i) a framework to test systems that generate mapping correspondences between relational schemas and ontologies, (ii) a scoring function to measure the quality of system-generated mappings, (iii) a variety of reference datasets and queries for benchmarking, which are called benchmark scenarios. These scenarios act as input to mapping generation systems where each scenario contains a source relational database (RDB), a target ontology (RDF) and a varying number of SQL-SPARQL query pairs. Figure 15 is a sample SQL-SPARQL test query pair.

```
name=Q29 (Amount of Conference Fees)
orderNum=330

sql=SELECT conference_fees.amount
FROM conference_fees

sparql=prefix rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
prefix : <http://conference#>
SELECT ?amount
WHERE {?fee rdf:type :Conference_fees; :amount ?amount}

categories=attrib,in-table
```

Figure 15: Sample SQL-SPARQL test query pair from an evaluation scenario (npd_atomic_tests)

Each SQL-SPARQL test query pair is designed by RODI to test one or more specific mapping challenge categories. These challenges are identified by "Tag IDs" by Pinkel et al. [5] . Table 29, which lists all the tags and the description of the mapping challenge. These challenges described by RODI encompass the requirements stated in Section 3.3 (also by RODI).

Table 29: Category tags used in SQL-SPARQL query pairs with the brief description. Each of these tags tests a specific or a combination of mapping challenges

Mapping Challenge	RODI Mapping Challenge Tag IDs
Matching class	*class*
Matching datatype property	*attrib*
Matching object property	*link*
1:1 table-class match	*1-1*

n:1 table-class match	*n-1*
1:n match table-class match	*1-n*
Requires n UNIONs to build match (a form of 1:n table-class match with explicit n)	*union-n*
Test on entities that should comprise sub class entities (form of 1:n)	*superclass*
Datatype match that finds the data value in the same table that also defines the related entity	*in-table*
Datatype match that finds the data value in a table other than the one that defines the related entity (requires joins)	*other-table*
Object property match that finds both related entities in the same table (1:1 or denormalized)	*Path-0*
Object property match that finds related entities in two tables that can be directly joined (single JOIN) or joined through one intermediate table (two JOINs). Note: NPD uses join-1, join-2 to denote the same aspect	*path-1,path-2*
Object property match that requires n JOINs (n > 2) to connect the tables that define entities on both sides. Note: NPD uses join-n to denote the same aspect	*path-n / join-n*
Additional tag for all queries that are tagged path-n, with any n > 1 (denotes multi-hop JOIN of any length)	*path-X*
Type filtering required due to denormalization	*denorm*
JOINs without leading foreign keys	*no-fk*

RODI provides both a jar file and a java source code on their GitHub[33]. The code uses several common, open source libraries[34] such as Apache Commons, PostgreSQL JDBC connector and Sesame.

5.1.3. Scenarios

This section first lists the rationale for selecting *cmt_renamed*, *cmt_structured*, *conference_nofk* and *npd_atomic_tests* as scenarios for evaluating Milan. This is done by listing the characteristics about the scenarios such as database size, ontology size, number of query-tests and various mapping challenges each of the scenarios contain. This is followed by a description of *cmt_renamed*, *cmt_structured* and *conference_nofk* scenarios which constitute the conference domain, and then the *npd_atomic_tests* scenario which constitutes the Norwegian Oil & Gas domain.

[33] https://github.com/chrpin/rodi

[34] Table 1 of the RODI manual shows a list of libraries that are being used.

The four input RODI scenarios that were selected to be used in the evaluation were: *cmt_renamed, cmt_structured, conference_nofk* and *npd_atomic_tests*. These scenarios have been selected based on criteria such as different domains, database size & complexity, variations in ontology modelling style, the presence of cardinality information and semantic expressivity of ontologies. Different domains allow evaluation of the lexical matching scope of Milan. For example, entity labels in the oil & gas domain such as (*wellbore_dst* to *Drill_Stem_Test_Wellbore)* can be less intuitive than the labels such as *(conference_paper* to *Conference_Paper_Submission)* in conference domain. Size of database and ontology are a good estimate of the complexity for automatic mapping correspondence generation. Variations in modelling patterns test that the diversity included in the scenarios cover the mapping challenges discussed in Table 29 above. Table 30 describes each of the scenarios in terms of domain, database/ontology size, number of test queries and the distinctive mapping challenges addressed.

Table 30: Description of benchmarking scenarios used in the experiment

Scenario Name	Domain	Database Tables	Database FK/PK	Ontology Classes	Query Tests	Distinctive Mapping Challenges (Tag ID)
cmt_renamed	Conference	48	69	23	29	*1-1, other-table*
cmt_structured	Conference	64	52	23	29	*path-1, path-2, path-n, n-1, in-table*
conference_nofk	Conference	60	0	59	39	*no-fk, n-1*
npd_atomic_tests	Oil&Gas	70	100	300	439	*1:n, union-n, path-1, path-2, path-n, join-n*

Table 31 below lists the number and the fraction of query-tests for each mapping challenge category (Tag ID) found in each scenario. Each cell is coloured to rank the datasets in the order of fractional dominance of the particular Tag ID e.g. 0.49 of the total queries in *npd_atomic_tests* tests in-table mapping challenge, which is the highest amongst all other four scenarios. This is followed by *cmt_structured* at

0.38, *conference_nofk* at 0.33 and lastly *cmt_renamed* having only 0.24 of its total queries under this challenge.

Table 31: Number (No.) and fraction (Fr.) of query-tests for each mapping challenge across four scenarios. The colour ranks the scenarios by the dominance of mapping challenge by fraction.

Tag IDs	cmt_renamed		cmt_structured		conference_nofk		npd_atomic_tests		Total
	No.	**Fr.**	**No.**	**Fr.**	**No.**	**Fr.**	**No.**	**Fr.**	**No.**
class	12	**0.41**	12	**0.41**	16	**0.41**	134	0.31	174
attrib	11	0.38	11	0.38	15	0.38	213	**0.49**	250
link	6	0.20	6	0.20	10	**0.26**	92	0.21	114
1-1	11	0.38	7	0.24	9	0.23	439	1	466
n-1	0	0	5	0.17	7	**0.18**	0	0	12
union-0	0	0	0	0	0	0	263	**0.6**	263
union-n	0	0	0	0	0	0	176	**0.4**	176
join-0	0	0	0	0	0	0	424	**0.97**	424
join-n	0	0	0	0	0	0	15	**0.03**	15
in-table	7	0.24	11	0.38	13	0.33	213	**0.49**	244
other-table	4	**0.14**	0	0	2	0.05	0	0	6
path-1,path-2	2	0.07	5	**0.17**	6	0.15	0	0	11
path-n	4	0.14	4	0.14	2	**0.05**	0	0	10
no-fk	0	0	0	0	10	**0.26**	0	0	10
INDEX ***Increasing order of rank***			**1**	**2**	**3**	**4**			

Conference Domain

The conference domain was chosen since (i) it is complex enough for realistic testing, and (ii) it has been successfully used in other benchmarks before (e.g. [55,12,7]). This experiment uses three different variants of scenarios for this domain, varying in size and complexity, each of which all built around the CMT ontology comprising of 23 classes and 77 properties with varying additions in *conference_nofk* scenario which has 59 classes. The corresponding databases vary in size between 48 tables/85 columns to 64 tables/125 columns. Number of query tests in each scenario range from 29 to 39. The conference ontologies in this benchmark are provided by the Ontology Alignment Evaluation Initiative (OAEI) ([64], [65], [66]) and were originally developed by the OntoFarm project.

The conference domain scenarios used in this experiment are as follows:

cmt_renamed: This scenario specialises in providing naming conflict challenges. Ontology designers typically consider different naming schemes than database architects, even when implementing the same (verbal) specifications [5]. Those differences include longer vs. shorter names, "speaking" prefixes, human-readable property IRIs vs. technical abbreviations (e.g. "*hasRole*" vs. "*RID*"), camel case vs. underscore tokenization, a preferred use of singular vs. plural, and others.

cmt_structured: This scenario specialises in providing restructured hierarchies challenges. It tests the critical structural challenge of identifying the different relational patterns to model class hierarchies, which, among others, includes the challenge to correctly build n:1 mappings between classes and tables.

conference_nofk: This scenario specialises in evaluating the challenge arising from databases with no foreign key constraints. In such a scenario, mapping tools must determine the correct join paths to connect tables that correspond to different entity types.

Oil & Gas Domain

This is another set of RODI scenarios based around a large and complex database. The scenario *npd_atomic_tests* was included as an example of a database and ontology currently in use in the real world, in the oil and gas domain, The Norwegian Petroleum Directorate (NPD) FactPages [49]. Its relational database contains a relatively complex structure (70 tables, ≈1,000 columns and ≈100 foreign keys). The database is constructed from a publicly available dataset containing reference data about past and ongoing activities in the Norwegian petroleum industry, such as oil and gas production and exploration. The corresponding ontology contains ≈300 classes and ≈350 properties. A total of 439 such queries have been compiled in the scenario *npd_atomic_tests* to cover all of the non-empty fields in the database. A specific feature resulting from the structure of the FactPages database and ontology is a high number of 1:n matches i.e., concepts or properties in the ontology that require a UNION over several relations to return

complete results. 1:n matches as a structural feature can, therefore, best be tested in the *npd_atomic_tests* scenario.

This ends the discussion of RODI benchmark scenarios used in the evaluation of Milan. Next the quality metric used in the experiment is discussed.

5.1.4. Quality Metric

The metric used to evaluate each SQL-SPARQL test query is the F-measure, which is the mean of precision and recall. If the result set from the SPARQL query using the mappings is res and the reference result from the SQL query is ref, then:

$$precision = \frac{|res| - unmatched|res|}{|res|} \quad (1)$$

$$recall = \frac{|ref| - unmatched|ref|}{|ref|} \quad (2)$$

where $unmatched|res|$ and $unmatched|ref|$ are the tuples that could not be matched across the SQL and SPARQL queries. The per-query F-measure $\bar{X}_{test}$ is the mean of $precision$ and $recall$ recorded for a single query-test. Figure 2 shows a sample evaluation report indicating the score ($\bar{X}_{test}$), $precision$ and $recall$.

```
['Q01(Persons)': score = 1.0; precision = 1.0, recall = 1.0]
['Q02(1st Authors)': score = 1.0; precision = 1.0, recall = 1.0]
['Q03(Co-Authors)': score = 1.0; precision = 1.0, recall = 1.0]
['Q04(Conferences)': score = 1.0; precision = 1.0, recall = 1.0]
['Q06(Reviewers)': score = 1.0; precision = 1.0, recall = 1.0]
['Q07(Documents)': score = 1.0; precision = 1.0, recall = 1.0]
['Q08(Papers)': score = 1.0; precision = 1.0, recall = 1.0]
['Q09(Abstracts)': score = 1.0; precision = 1.0, recall = 1.0]
['Q10(Reviews)': score = 1.0; precision = 1.0, recall = 1.0]
```

Figure 16: Screenshot of RODI benchmarking framework evaluation report showing score, precision and recall for each SQL-SPARQL test query pair

The RODI mapping quality evaluation metric also generates the aggregated score, precision and recall metric '*ALL (AVG)*' which is the mean of all scores, precision and recall respectively over all tests in a scenario. In addition to this, mean of score, precision and recall belonging to each mapping challenge category of Table 1 is also reported. Figure 3 shows a sample evaluation report indicating the total mean and category-wise mean scores, precision and recall.

```
['All (AVG)': score = 0.75; precision = 0.75, recall = 0.75]
['path-3': score = 0.33; precision = 0.33, recall = 0.33]
['path-2': score = 1.0; precision = 1.0, recall = 1.0]
['path-1': score = 1.0; precision = 1.0, recall = 1.0]
['other-table': score = 0.5; precision = 0.5, recall = 0.5]
['1-1': score = 0.91; precision = 0.91, recall = 0.91]
['attrib': score = 0.72; precision = 0.72, recall = 0.72]
['in-table': score = 0.85; precision = 0.85, recall = 0.85]
['superclass': score = 1.0; precision = 1.0, recall = 1.0]
```

Figure 17: Screenshot of RODI benchmarking framework evaluation report showing means of the score, precision, recall for all queries and for each category

5.1.5. Experiment Methodology

This experiment involves four scenarios, each constituting a relational database, target ontology and a set of SQL-SPARQL test query pairs. The evaluation involves comparing the performance of Milan with seven other state-of-the-art systems, namely: BootOx, IncMap, Ontop, Automatic Mapping Generation for OBDA, D2R, MIRROR and COMA.

The general architecture of the experiment is shown in Figure 18. In general, the experimental methodology involves 3 steps: first, loading the database file provided by RODI in a RDBMS and ontology into a triplestore; second, running Milan to generate a set of correspondences from the RDB and ontology; third, using the quality metric to evaluate the quality of correspondences generated by Milan. This includes scoring Milan's overall performance and performance in individual mapping challenge categories (Tag IDs).

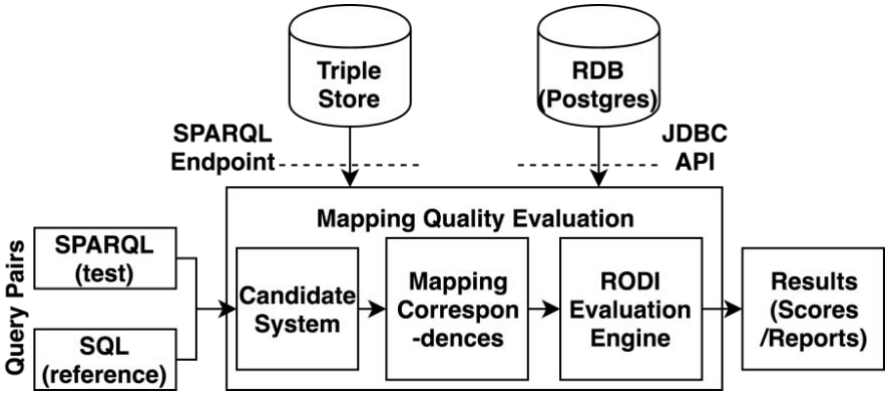


Figure 18: Architecture of experiment evaluation

In a bit more detail a run through of the experimental process follows, considering the *cmt_renamed* scenario. The same was repeated over the other three scenarios.

Step 1: Load Data

The file dump.sql[35], containing the relational database is hosted on a PostgreSQL database. The ontology.ttl file contains the T-box of the ontology, which is hosted on Apache Fuseki triplestore.

Step 2: Generate Correspondences

Milan accesses the PostgreSQL database via the JDBC API and the Fuseki triplestore via a SPARQL endpoint and executes its algorithm to generate mapping correspondences.

Step 3: Correspondence Evaluation

In the present form of this work, the evaluation of correspondences is done manually in two phases. First, establishing the gold standard of correspondences by inspecting the query pairs. This phase is independent of the mapping correspondences generated from Milan. Hence it is done before evaluating Milan's mapping correspondences. Second Milan's output was compared with the gold standard mapping correspondences inferred in the first phase. In this evaluation, the

[35] https://opengogs.adaptcentre.ie/mathurs/Milan/src/master/Da

quality metric is used to score the mapping correspondences for each query-test. The scores for each query-test are then aggregated to synthesize a RODI report for comparison with other state-of-the-art systems.

For each phase, three classifications exist which are based on the three non-overlapping mapping challenges of class tag (Class-Table), link tag (FK/PK–Object Property) and attribute tag (Column–Data Property). The rest of this section illustrates the evaluation methodology for these three classifications, each discussing both phases of correspondence evaluation.

Table-Class Tags

For this classification, the RODI SQL-SPARQL query-test pair provides both the source table(s) and target class(es). Hence the first phase will involve inferring one or more tables from the relational database that are expected to match one or more classes from the ontology. Table 32 (B) is an example of establishing gold standard for 1-1 class-table match and Table 34 (B) is an example of n-1 class-table match.

Algorithm 1 of Milan which detects class-table relationship provides table-class mapping correspondences. This results in consolidated correspondences which include 1-1, 1-n, n-1, m-n class-table mapping correspondences. Hence results from this algorithm are used to evaluate Milan's performance in detecting class-table (class tag) correspondences. Table 33 and Table 35 show partial results from Algorithm 1 which are used to evaluate using the SQL-SPARQL query-tests of Table 32 (A) and Table 34(A).

Table 32, provides an example of a query evaluation (Q1) for *the cmt_renamed* scenario. The SQL query counts the number of tuples in table persons, while the SPARQL query counts the number of URI which is an instance of Person class. Thus, by manual inspection of the query pair this correspondence has been identified.

When evaluating Milan outputs, if the mapping correspondence persons-Person exclusively exists in the Milan output, then score, precision and recall are all marked as 1.0 for this query test evaluation. Exclusively in this context means

that while *persons-Person* exists, there should be no other correspondence either for table *persons* or for class *Person*. Table 33 confirms that Milan indeed detected the correspondence correctly, without any other correspondence for either the table or the class.

Table 32: Example of first phase of correspondence evaluation. Establishing gold standard for 1:1 table-class correspondence. (A) shows the RODI SQL-SPARQL test query pair along with (B) gold standard table-class correspondence inferred from (A)

RODI Query-Test Pair (A)	Gold Standard for Table-Class Correspondences (B)
SQL= **SELECT COUNT**(*) **FROM** persons	**persons** ↔ **Person**
SPARQL = SELECT (**COUNT**(?per) as ?count) **WHERE** {?per rdf:type :Person}	

Table 33: Example of the second phase of correspondence evaluation. Partial Snapshot of the consolidated class-table correspondences produced by Milan indicating the existence of persons-Persons correspondence

Table	Table Condition	Class
authors	NA	Author
co-authors	NA	Co-author
conferences	NA	Conference
documents	NA	Document
persons	**NA**	**Person**

Table 34 and Table 35 demonstrates a more difficult case of n:1 table-class matching, where subpart of tables *facility_moveable* and *facility_fixed* based on a column condition of *fclKind = 'JACKET 4 LEGS'*, the union of both form the *Jacket4LegsFacility* class. Hence to correctly detect the mapping correspondences, Milan must exclusively identify both the tables for the given class with the appropriate column condition. Exclusively in this context means, that either of the two tables (*facility_moveable, facility_fixed*) should not match to a class other than *Jacket4LegsFacility* by the same table condition. Table 35 shows that both the

required correspondences are discovered by Milan hence correctly finding the required correspondences.

Table 34: Example of first phase of correspondence evaluation. Establishing gold standard for 1:n table-class correspondence. The example shows the (A) SQL-SPARQL query-test pair along with gold standard table-class correspondence inferred from (A) where tables facility_moveable and facility_fixed both under a column condition collectively form Jacket4legsFacility class.

RODI Query-Test Pair (A)	Gold Standard for Table-Class Correspondences (B)
SQL= **SELECT COUNT**(*) **FROM**(**SELECT** **CONCAT**('http://sws.ifi.uio.no/data/npd-v2/facility/', **CAST**(TABLE1.fclNpdidFacility AS CHARACTER VARYING)) **AS** x **FROM** facility_moveable TABLE1 **WHERE** (TABLE1.fclKind = 'JACKET 4 LEGS') **AND** TABLE1.fclNpdidFacility **IS NOT NULL** **UNION ALL** **SELECT** **CONCAT**('http://sws.ifi.uio.no/data/npd-v2/facility/', **CAST**(TABLE1.fclNpdidFacility AS CHARACTER VARYING)) **AS** x **FROM** facility_fixed TABLE1 **WHERE** (TABLE1.fclKind = 'JACKET 4 LEGS') **AND** TABLE1.fclNpdidFacility **IS NOT NULL**) AS T	**facility_fixed ∪ facility_moveable ↔ Jacket4LegsFacility** (where condition clause for Tables) **facility_fixed.fclKind = 'JACKET 4 LEGS'** **facility_moveable.fclKind = 'JACKET 4 LEGS'**
SPARQL= **SELECT** (**COUNT**(*) **AS** ?count) {?x a npdv:Jacket4LegsFacility }	

Table 35: Example of the second phase of correspondence evaluation. Partial Snapshot of the consolidated class-table correspondences produced by Milan indicating the exclusive existence of two tables facility_fixed and facility_moveable, each under column fclKind='JACKET 4 LEGS' matches to class Jacket4LegsFacility

Table	Table Condition	Class
Wellbore_exploration_all	wlbDiskosWellboreType= 'Initial'	InitialWellbore
facility_fixed	fclKind = 'JACKET 12 LEGS'	Jacekt12LegsFacility
facility_fixed	**fclKind = 'JACKET 4 LEGS'**	**Jacket4LegsFacility**
facility_moveable	**fclKind = 'JACKET 4 LEGS'**	**Jacket4LegsFacility**
facility_fixed	fclKind = 'JACKET 6 LEGS'	Jacekt6LegsFacility

Milan's opengog repository[36] comprises of a complete list of Table- Class correspondences.

FK/PK – Object Property Tags

For the FK/PK (Foreign Key/Primary Key)-object property cases, the RODI SQL-SPARQL queries provide the pair of tables which are related by one or more FK/PK relations and a corresponding pair of classes, which are related by an object property. The section presents examples of evaluation for link tag IDs. Table 36(A) shows the SQL-SPARQL test query pair. Table *paper* is linked to table *co_authors* via *co_author_paper* junction table. Table *paper_full_versions* is linked to table *paper* by a 1:1 table-table relationship, thereby inheriting the object and data properties of *paper*. Hence if *paper* is mapped to *co_author*, table pair *paper_full_versions-co_author* is inferred from property inheritance. Hence to evaluate this test query, Milan should be able to establish a correspondence between the object property *co-writePaper* and junction table *co_author_paper*. Table 37 shows the results from Algorithm 2, where tables *co_author* and *paper* are related by junction table *co_author_paper*, which corresponds to object property *co-writePaper*. Hence the evaluation results in 1.0 in both precision and recall.

Table 36: Example of first phase of correspondence evaluation. Establishing gold standard for junction table based FK/PK-object property correspondence. The example shows the (A) SQL-SPARQL query-test pair along with gold standard FK/PK-object property correspondence inferred from (A)

RODI Query-Test Pair (A)	Gold Standard for FK/PK-Object Property Correspondences (B)
SQL= **SELECT COUNT**(*) **FROM** paper_full_versions NATURAL JOIN papers JOIN co_author_paper ON papers.id=co_author_paper.pid JOIN co_authors ON co_author_paper.cid = co_authors.id	JT(Table1,Table2) ↔ OP(Class1, Class2) **co_author_paper (co_author-paper_full_version) ↔ co-writePaper (Co-author, PaperFullVersion)**
SPARQL= SELECT (**COUNT**(*) **AS** ?cnt) **WHERE** {?paper a :PaperFullVersion . ?author a :Co-author; :co-writePaper?paper }	

[36] https://opengogs.adaptcentre.ie/mathurs/Milan

Table 37: Example of the second phase of correspondence evaluation. Partial Snapshot of the consolidated FK/PK-object property correspondences produced by Milan. First column lists the pair of tables involved in the primary-foreign key relation, second column lists the label of the foreign key columns or the label of the junction table, third column lists the label of the matching object property. The fourth column describes the type of matching made by Milan.

Table-Table (PK,FK)	Join Con. (JT/FK_col)	Object Properties	Type
administrators, paper	accepted	acceptedBy	D2R
administrators, paper	rejected	rejectedBy	D2R
conferences, conference_members	conference	hasConferenceMember	D2R
co_authors, papers	co_author_paper	co-writePaper	IJ
co_authors, paper_full_versions	**co_author_paper**	**co-writePaper**	**i-IJ**

Column – Data Property Tags

For the column – data property correspondences, the SQL-SPARQL test queries provide the source "table.column" from the database and target data property and its range class. Table 38 is an example of evaluation for "attrib" Tag ID.

Table 38: Example of first phase of correspondence evaluation. Establishing gold standard for column-data property correspondences. The example shows the (A) SQL-SPARQL query-test pair along with gold standard column-data property correspondence inferred from (A)

RODI Query-Test Pair (A)	Gold Standard for Column-Data Property Correspondences (B)
SQL= **SELECT** persons.email **FROM** persons	**Persons.email** ↔ **Person.email**
SPARQL= **SELECT** ?email **WHERE** {?p rdf:type :Person; :email ?email .}	

Table 39 shows the correspondences produced by Milan (partial output). The correspondence between column *email* of table *Persons* and data property *email* of class *Person* has been correctly identified by Milan using Algorithm 3, that uses the

class-table correspondence between *Persons* and *Person* from Algorithm 1 as inputs.

Table 39: Example of the second phase of correspondence evaluation. Partial Snapshot of the consolidated column-data property correspondences produced by Milan. The table shows column email of table persons matches data property email of class Person

Table	Column	Data Property	Domain Class
persons	name	name	Person
persons	**email**	**email**	**Person**
conferences	name	name	Conference

5.2. Results

This section presents the results of the experimental evaluation of the quality of mapping correspondence generation of Milan. The results consist of tabulated F-measure values generated by testing using the RODI framework for third party systems and manually synthesized RODI reports generated by correspondence inspection for Milan (as described in the last section). Section 5.2.1 presents the F-measure for each scenario by Milan and the 7 other tested systems. The results of the other seven systems are based off published results. Section 5.2.2 shows the comparison of F-measure for the *cmt_renamed* scenario broken down into class (table-class), attrib (column-data property) and link FK/PK object property) tag ID classification. This is done only for *cmt_renamed* as results for these tags have not been published by authors of other systems compared in this evaluation. Section 5.2.3 presents plots that break down the mean F-measure for each mapping challenge constituted in the four scenarios.

5.2.1. Overall F-Measures

Table 40 shows the F-measure for each scenario on every tested system. This is the value obtained from the 'All (AVG)' tag of the generated evaluation report. It is calculated by taking the average of the F-measures generated for each query-test in the scenario (see 5.1.4). These results show that all systems are far from achieving

complete accuracy in automatically generating mappings. Milan outperforms the tested systems in this metric, for each of the four tested scenarios. Hence the hypothesis state in Section 5.1.1 is accepted.

Table 40: Comparison of overall per scenario F-measures. In each case the highest value observed is bolded. Based on this evaluation, § denotes the best and * the next best system in each scenario

Scenario	BootOx	IncMap	Ontop	MIRROR	COMA++	D2RQ	AutoMap - 4OBDA	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_nofks	0.33	0.41*	-	0.17	0.21	0.18	0.41*	**0.46**§
npd_atomic_tests	0.14	0.16	0.10	0	0.02	0.08	0.23*	**0.30**§

This table shows that in *cmt_renamed*, Milan performs 13.15% better than its next best system BootOx. Similarly, in *cmt_structured*, Milan performs 18.18% better than IncMap; in *conference_nofks,* Milan performs 12.19% better than IncMap and AutoMap4OBDA; in *npd_atomic_tests,* it performs 30.43% better than its next best AutoMap4OBDA. Trend is observed that the effectiveness of Milan becomes more evident with a rise in the complexity and size of the database to be mapped.

5.2.2. F-Measures under Class, Attrib and Link

This section splits the evaluation data down into three broad, non-overlapping mapping challenges addressed in order to get insight into the relative performance of Milan. These challenges are table-class, column-data property and FKPK-object property correspondences which are identified as class, attrib and link in tag ID of mapping challenges. Figure 19, is a grouped bar plot comparing the F-measure for *cmt_renamed* scenario produced by various tested systems. This plot does not present the results for the AutoMap4OBDA (A4MO) system, as the authors of A4MO have not been published these results and have not yet publicly released their code. It can be inferred from the plot that Milan performs as good as BootOx in detecting class-table correspondences, and both are have significantly better performance at addressing class tag ID mapping challenges than the other systems.

Milan performs significantly better under attrib tag ID (37% improvement) and link tag ID (34% improvement) correspondences than the next best. It is notable that many systems do not detect any correspondences (F-measure value of 0 in the plot) for attrib and link.

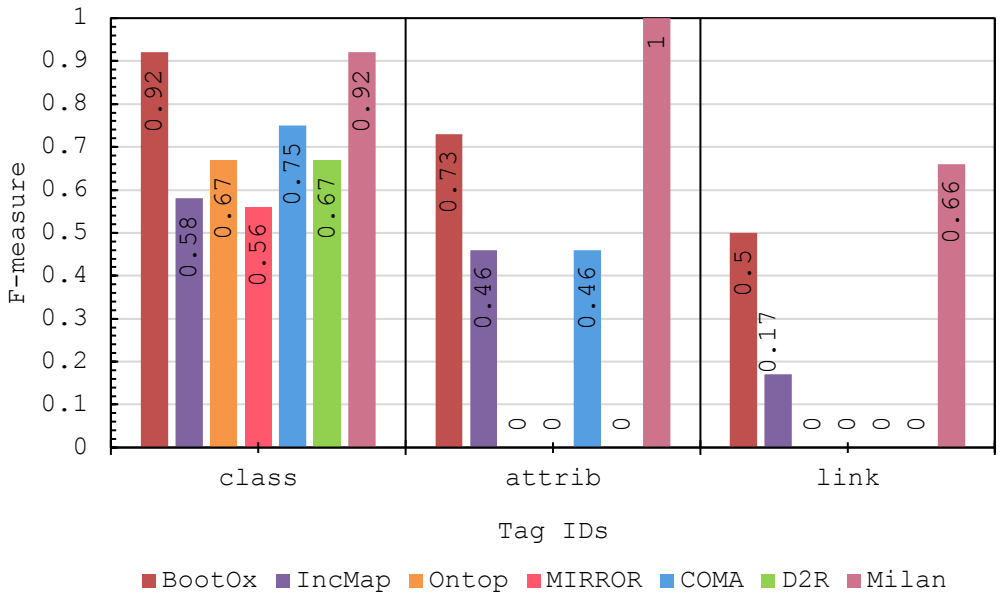


Figure 19: Grouped bar plot, comparing the F-measure across class, attrib and link tag IDs for *cmt_renamed* scenario.

5.2.3. F-Measures for each mapping challenge across scenarios

This section provides an even more detailed look at the data. It considers all the tag IDs belonging to the scenario to infer the strengths and weaknesses of Milan. Unfortunately, the other systems have not published evaluation data at this granularity, so it is not possible to do direct comparisons here. However, this data will help us explore the factors that contribute to Milan's favourable results in the previous comparisons. The plots here illustrate the individual mean F-measure per mapping challenge in the four scenarios – *cmt_renamed*, *cmt_structured*, *conference_nofks*, *npd_atomic_tests*.

Figure 20, describes the bar plot of average of F-measures aggregated over mapping challenge categories (tag ID) for *cmt_renamed* scenario. It is important re-emphasize that one query-test can contain one or more tag IDs.

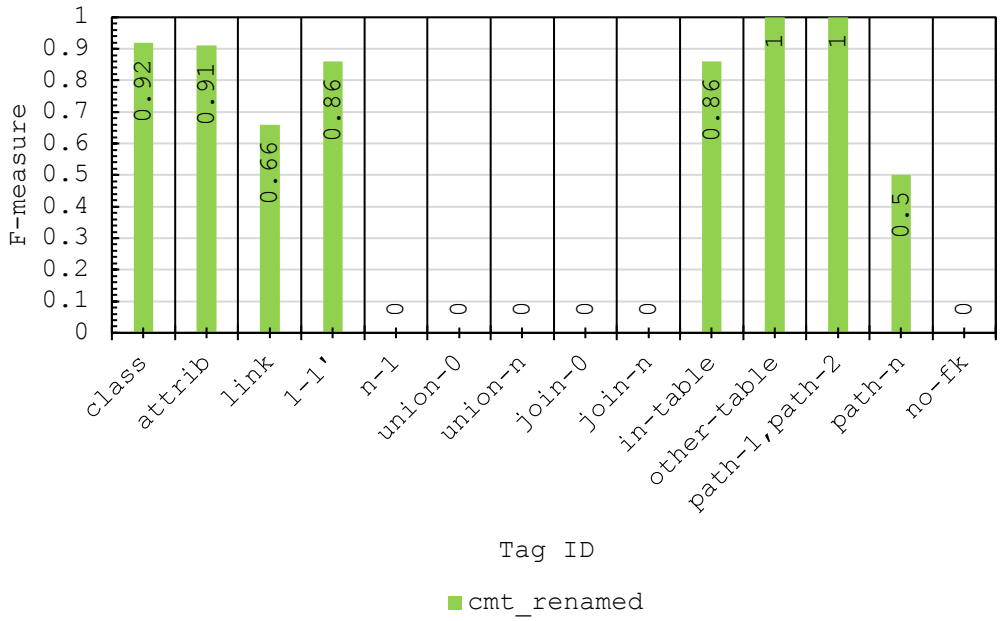


Figure 20: Bar plot representing F-measures for Milan across various mapping challenges (tag IDs) in the cmt_renamed scenario

In general, the scores are impressive with almost all mapping challenge category scoring more than 0.5. For four of the tags (*path-2*, *path-1*, *othertable* and *superclass*) Milan achieved a perfect score by finding all the correspondences. However, it scored a 0 on path-4, which contained one query (Q46). However, this occurred due to a false positive result in Table-Class detection. Milan incorrectly matched class *ProgramCommitteeMember* to table *program_committee_members* instead of (*ProgramCommitteeMember, pc_members)* match. Since Q46 was based on this correspondence, Milan score 0 on this query-test.

The "other-table" tag cases involve 1-1 table-table relationships, but they are sometimes hidden under multiple indirections in the database e.g. Query pair 32 of *cmt_renamed* where parent table *papers* is linked to child table *paper_full_versions* by a 1:1 relationship. *paper_full_versions* inherits column *paper_id* of *papers* and matches it to data property *paperID* of its corresponding class *PaperFullVersion.*

Another distinctive tag is path-3, which involves the presence of junction tables in relational database as discussed in Table 7. Milan successfully identifies

67% of these correspondences e.g. Query pair 43 of *cmt_renamed*, object property *hasBeenAssigned* is the predicate and classes *Reviewer* and *PaperFullVersion* are the subject and object respectively. The corresponding pair of tables *reviewer* and *paper_full_versions* are not directly connected by FK/PK. They are linked via junction table *paper_reviewer*.

The mean F-measure of 0.92 for tag "Class" is due to one wrong class-table correspondence where Milan matches class *ProgramCommitteeMember* to table *program_committee_members* while the true correspondence of the class is to table pc_members. This mismatch affects score of tags 1-1, path-X, path-4 and link.

Figure 21, presents a plot of F-measures for queries belonging to the various mapping challenges (tag ID) present in the *cmt_structured* scenario. In general, the overall performance seems lower than in the *cmt_renamed* scenario.

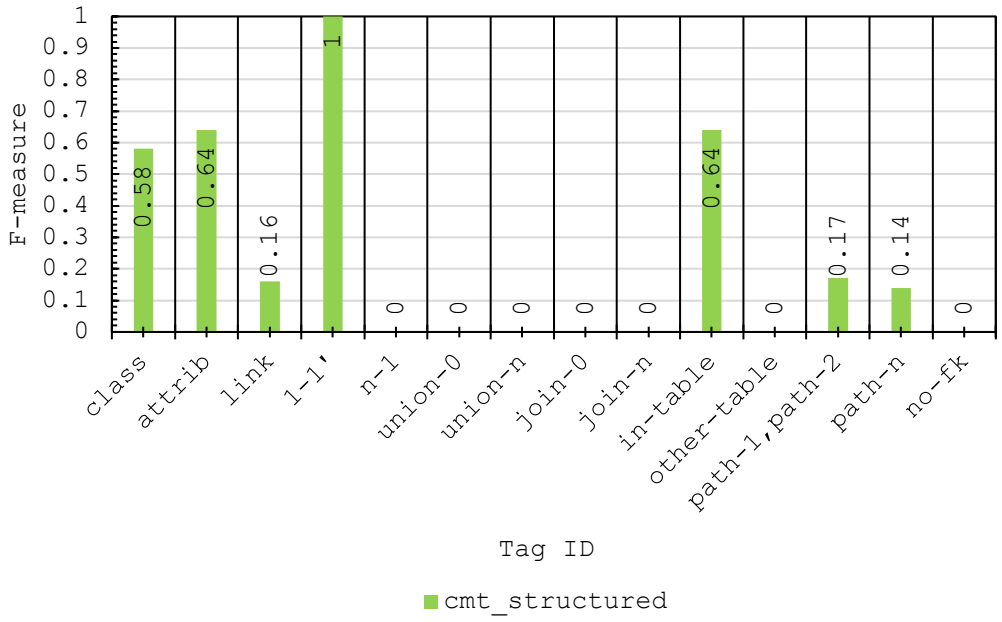


Figure 21: Bar plot representing F-measures for Milan across various mapping challenges (tag IDs) in the cmt_structured scenario

Milan is able to correctly detect all 1-1 class-table correspondences however it fails to detect the n:1 class-table correspondences present. Each of the 5 queries belonging to this mapping challenge split columns based either on Boolean variables or non-lexical categorical variables e.g. In *cmt_structured* scenario, class

Author is linked to table *Person* where column *is_Author* is *true*; class *PaperAbstract* is linked to table *Paper* where column *TYPE* is *2*.

Figure 22, describes the plot of mean F-measures for queries belonging to the mapping challenges in the *conference_nofks* scenario. In general, the mean F-measures show that while Milan detects more than 70% of the 1-1 class-table and data properties-column (attrib) correspondences, it is unable to detect any object property-FK/PK correspondences in this case, therefore having a 0 mean F-measure for the tags: link, path-1/2/3/X, no-fk . Low scores in the class tag is due to the failure in correctly detecting n:1 class-table correspondences, facing same concern as the previous scenario (Figure 21).

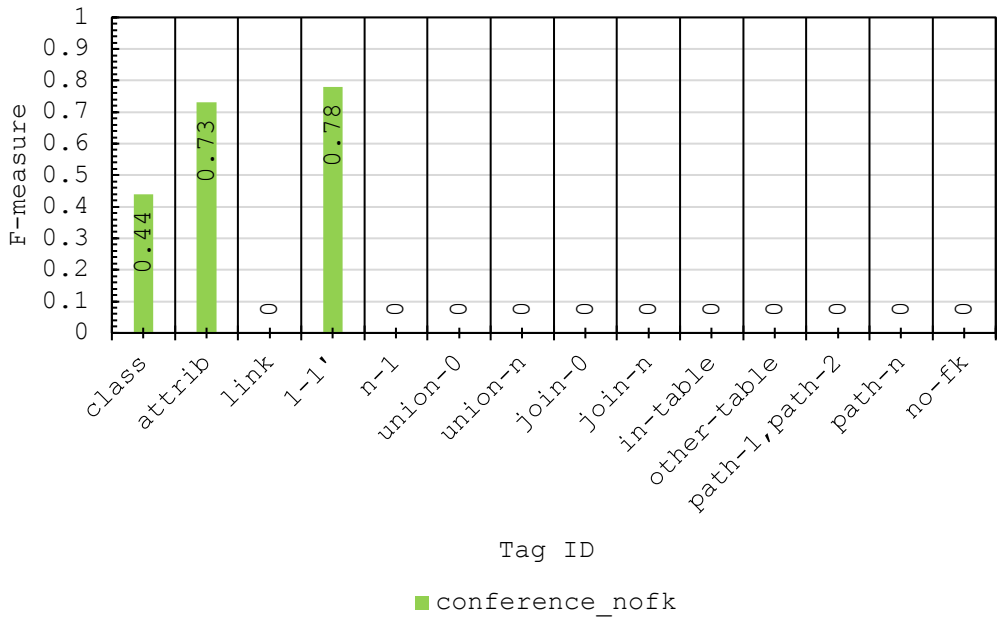


Figure 22: Bar plot representing F-measures for Milan across various mapping challenges (tag IDs) in the cmt_structured scenario

Figure 23, describes the plot of F-measure for queries broken down by mapping challenge type in the *npd_atomic_tests* scenario. In general, the mean F-measures shows the ability of the holistic approach of Milan to detect different categories of correspondences. It detected 55% class-table correspondences, of which many are n:1 class table matches.

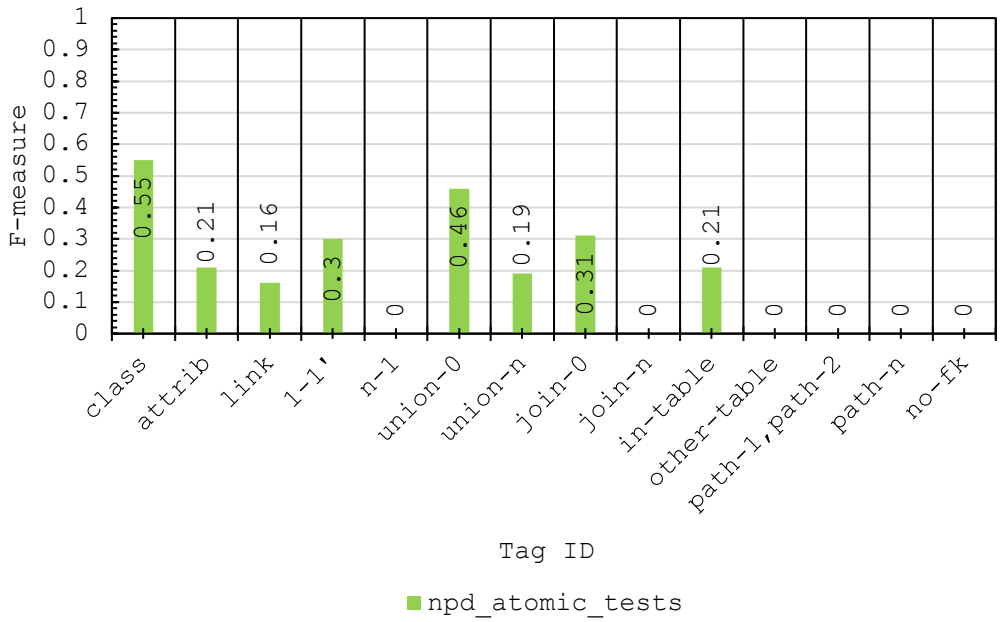


Figure 23: Bar plot representing F-measures for Milan across various mapping challenges (tag IDs) in the npd_atomic_tests scenario

n:1 class table matches are detected by tag Union-0 and Join-0 e.g. Query pair (orderNum=106) of *npd_atomic_tests* where class *ReClassToTestWellbore* is linked to table *wellbore_exploration_all* with column *wlbStatus* as '*RE-CLASS TO TEST*'. Similarly, numerous classes are linked to table *wellbore_exploration_all*. These are correctly detected by Milan by the means of column splitting.

1:n class-table matches are detected by tag Union-1, a distinctive tag present in this scenario. 11% of these are correctly detected by Milan e.g. Query pair (orderNum=58) of *npd_atomic_tests* where class *InitialWellbore* is linked to tables *wellbore_development_all* and *wellbore_exploration_all* with each having the condition to their column *wlbDiskosWellboreType* as *'initial'*.

5.3. Analysis

This section analyses the results of the evaluation performed to quantify the quality of the mapping correspondences generated by Milan.

The section first lists the key findings of the evaluation, describing the scenarios and mapping challenges where Milan's performance was good, pointing

out parts and sub-parts of Milan's methodology responsible for it. Similarly, for the weak performance. This section concludes with a discussion on the ambit of the evaluation methodology.

It must first be observed that Milan outperformed the other tested systems on overall F-measure across all four scenarios. It observes an improvement from the next best system in each scenario by an average of approximately 18.4%. An observed trend shows that Milan's extent of improvement from other system increases with an increase in complexity and size of relational database and ontology.

Success at Mapping Challenges

Milan performs equally well as the other systems in detecting class-table correspondences in trivial scenarios such as *cmt_renamed* but is seen to perform significantly well in scenarios offering complex mapping challenges such as Union-0 and Union-1 that involve 1:n and n:1 tag ID mapping challenges. This is by virtue of Milan's holistic approach to label matching and leveraging class-table patterns. Milan's Algorithm 1 addresses the above mapping challenges using the discovery of column splitting in tables to match to potential sub-classes and sibling classes. In the *npd_atomic_tests* scenario, Milan correctly detected 74 query-tests correctly under the 'class' tag. Milan's contribution to the overall score from the 'class' tag (.17) itself is higher than the total scores achieved my all systems except A4MO. It is able to detect complex table-class patterns as is evident from its performance over the class, join, union tags. This is credited to Milan's approach to detect union-n, join-0 and union-0 tag ID that involve detecting 1-n table-class correspondences.

In the *cmt_remaned* scenario, it can be seen from Figure 21 only 3 out of 7 seven tested systems haves a non-zero score for the 'link' tag ID. Milan outperformed in this tag against the compared systems. Within the 7 query-test for 'link' mapping challenges, 2 query-tests belong to path-1,2 bearing a f-measure of 1, 4 belong to path-n bearing a f-measure of 0.5 and 1 belongs to other-table which also had a f-measure of 1. These high scores are credited to is credited to 1:1 table-table matching, where the FK/PK relationships are inherited by the child table from

the parent table. 1:1 table-table matching challenge comprised of 34% of the primary/foreign key relations in *cmt_renamed*.

A dominant reason for improved data property detection is the inheritance of properties in the 1:1 table-table relationship pattern. In addition to this, Algorithm 3 uses annotation properties such as rdfs:label to match to the column. Using the annotation properties is the only cause for a marginal improvement (12%) in scores in *conference_nofk* scenario. Although the evidence for the strategy of datatype partitioning cannot be quantified as the alternate methodology was not tested, but it reduces false positive lexical matches by leveraging the datatype information.

Failure at Mapping Challenges

Although many 1:n table-class relationships observing the column splitting pattern are present in *cmt_structured*, Milan failed to detect them correctly. Algorithm 1 was not so successful here. 6 columns were split based on binary values. For example, 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 an undetected correspondence where the matching class was a sub-sub-class.

Milan couldn't capture object property mappings when object properties in the ontology do not have explicit rdfs:range/domain or owl:unionOf assertions (for example when OWL restriction classes are used). Additionally, Milan failed to detect any object property-FK/PK correspondences in the *conference_nofks* scenario due to its heavy dependence on the availability of FK/PK meta-data in relational databases. There is no FK/PK relationship in *conference_nofks* scenario. Milan was also unable to detect sub-sub-class relationships, in table-class match.

The tables of the *conference_nofk* scenario database has columns which have a Boolean datatype while the matching data property is a string. Milan is currently unable to match these cases, thereby missing on some of the data properties.

Conclusion

Milan's algorithms holistically find correspondences by leveraging relational to ontology matching patterns. The intelligent column splitting method to detect n:1 matches, leveraging junction tables and column&key inheritance, Hungarian algorithm for matching optimization majorly contributes to the success in this experimental evaluation over other tested systems.

The scenarios provided by RODI are designed to be realistic, involving real cases. However, the RODI benchmarking falls short on stress testing. It does not offer datasets where table labels are complex alpha numerals and do not hold any lexical resemblance to class labels. Therefore, it will be interesting to see how Milan performs on obtuse datasets. For example, a table-class match where the labels are *abc_123* and *xyz_001*. It would be insightful to see how Milan performs on such cases. It must be noted that Milan has been designed keeping such cases in mind. Hence while it might not be able to match such obtuse labels, it will not throw false positives either.

Although Milan performs better than other systems, it has great scope for improvement. This is evident by looking at the average f-measure of 0.535 across the four scenarios. Table 41 below lists the number of query-tests, f-measure for each tag ID along with the scope for improvement in the last column. The number of '*' in the last column denotes the extent of the scope for improvement, where the higher number of '*' means greater scope of improvement.

Table 41: Table showing the number of query-test for each Tag ID and the evaluation score achieved by Milan.

Tag IDs	cmt_renamed		cmt_structured		conference_nofk		npd_atomic_tests		Impr.
	No.	**score**	**No.**	**score**	**No.**	**score**	**No.**	**score**	**(Ind)**
class	12	.92	12	0.58	16	0.44	134	0.55	**
attrib	11	0.91	11	0.64	15	0.73	213	0.21	*
link	6	0.66	6	0.16	10	0.00	92	0.16	**
1-1	11	0.86	7	1	9	0.78	439	0.30	*
n-1	0	0.00	5	0.00	7	0.00	0	0.00	***
union-0	0	0.00	0.00	0.00	0	0.00	263	0.46	*
union-n	0	0.00	0.00	0.00	0	0.00	176	0.19	**
join-0	0	0.00	0.00	0.00	0	0.00	424	0.31	**
join-n	0	0.00	0.00	0.00	0	00.00	15	0.00	***

in-table	7	0.86	11	0.64	13	0.00	213	0.21	**
other-table	4	1.00	0.00	0.00	2	0.00	0	0	*
path-1,path-2	2	1	5	0.17	6	0.00	0	0	*
path-n	4	0.50	4	0.14	2	0.00	0	0	**
no-fk	0	0	0	0	10	0.00	0	0	***

5.4. Chapter Summary

This chapter described an experimental evaluation of Milan that tested the quality of Milan's mapping correspondence generation. The chapter introduced the RODI benchmark approach, various scenarios (datasets) used in this experiment, evaluation metric, experimental methodology. It then described the results, comparing them with other tested systems. The chapter finally concludes with the analyses of the evaluation result of the experiment

6. Conclusion

This chapter first provides general findings on the research question (Section 6.1), then it evaluates the progress towards the three research objectives (Section 6.2). Section 6.3 discusses potential future work and Section 6.4 presents final remarks.

6.1. Evaluation of Achievement of Research Objectives

Milan, a relational-to-ontology mapping system has been designed, developed and evaluated to address the research question – *To what extent can an automatic RDB to RDF mapping generation technique, based on semantic, lexical and structural analysis of both a source relational database and a target ontology result in quality mapping correspondences?*

Milan on an average, produced a mapping quality score of 53.5. This meant that it correctly generated 53.5% of the ideal mappings required. This performance is an improvement by 18.4% from the next best state-of-the-art system. This was achieved using lexical, semantic and structural analysis. Milan as part of its lexical analysis performed label matching, where complex labels of the entities of relational databases were matched to the labels of concepts of ontologies. Structural analysis involved matching the schema to the relational database to the schema of the ontology such as matching FK/PK relationship pair to object properties, non-key columns to data properties. Semantic analysis involved bridging the inter-model mismatch present between relational databases and ontologies. This included reasoning over OWL constructs to match two object properties which are inverse of each other to a FK/PK relationship pair.

.

6.2. Evaluation of Achievement of Research Objectives

This section analyses the achievements of this thesis for each of the research objectives associated to the research question of this thesis. Section 6.2.1 discusses the first research objective (RO1), which analyses the state the art in relational-to-ontology mapping patterns as well as state-of-the-art, automatic relational-to-ontology systems. Section 6.2.2 discusses the second research objective (RO2),

which analyses the design and development of Milan. This was done carefully considering the requirements and the gaps identified from the existing state-of-the-art systems (sec. 3.3). Section 6.2.3 discusses the third research objective (RO3), which analyses the rigour of the evaluation methodology, the evaluation results for Milan and other compared state-of-the-art systems.

6.2.1. RO1: Analyse the state-of-the-art in relational-to-ontology mapping (RDB2RDF) patterns and automatic relational-to-ontology mapping generation systems

The state-of-the-art survey comprised of two parts. First, reviewing work in the area of relational-to-ontology patterns and inferring the requirements for automatic relational-to-ontology mapping systems. Second, reviewing the capabilities of state-of-the-art automatic mapping generation systems.

The review of the challenges faced in the context of relational-to-ontology mapping patterns. In the process of reviewing more than 15 papers, the most structured previous review of the literature is by Pinkel et al. [5], which cites Batini et al. [24] to describe three classifications of challenges in relational-to-ontology mapping - naming conflicts, structural heterogeneity and semantic heterogeneity. Figure 8 shows examples of these challenges and Table 7 lists the automatic mapping system requirements based on these challenges. In Table 7, three requirements (R1, NC1 and NC2) are new additions to Pinkel et al.'s Table [5]. These add to the requirements and increase the completeness of specification of the naming conflict challenge. This has been discussed in Section 3.3. The section also finds that while the requirements of Pinkel et al. [5] are the most widely accepted and informative compilation of automatic relational-to-ontology systems challenges, the requirements could have been better elaborated and supported via substantive example.

The review of existing systems described seven state-of-the-art systems. It also compared these systems to the system requirements inferred in Section 3.3. The general outcome of this comparison is that most systems are far from fulfilling all requirements. All systems perform well up to the level of simple Direct

Mappings, where they can match a table to a class, a FK/PK relation to an object property and column to a data property. Section 3.4 shows that amongst the seven approaches, most systems only fulfil a fraction of the mapping requirements, hence justifying the need for research and scientific work in this area. This analysis, eventually paved as stepping stones to design Milan.

6.2.2. RO2: Design and develop a multi-level algorithm-based system that employs lexical, structural and semantic analysis to automatically discover correspondences between RDB and RDF

This research objective was fulfilled by designing Milan, which is a system to automatically generate mapping correspondences between a source relational database and target ontology. Milan performs lexical, structural and semantic analysis using 3 main processes and 6 sub-processes. Chapter 4 discusses Milan's 3 main processes - Table-Class correspondence identification, FK/PK-Object Property correspondence identification and Column-Data Property correspondence identification. Milan uses a relational database and an ontology (T-box only) as inputs to produce a set of mapping correspondences as an output. 2 out of 6 sub-processes, namely, label matching and combinatorial optimization use generic algorithms previously developed and used for varied applications in computer science, economics and mathematics. 4 out of 6 sub-processes are Milan-specific sub-processes namely, Datatype Disambiguation, Property Inferencing, Junction-Table Detection and Column Splitting. They are discussed in detail in Section 4.4. Each process and the sub-processes associated with are encapsulated as UML process diagram and algorithms.

Then design of Milan (Chapter 4) provides innovative strategies divided into three main processes and six sub-processes, to addresses different relational-to-ontology patterns in the context of automated mapping generation. For example, consider option 1 of Figure 8. The table Persons matches two classes, *Author* and *Reviewer*. Hence Table-Class correspondence identification process should be able to detect column *type* as a column to split the table based on the values – *author* and *reviewer*. In order to do so, the process included an algorithm to scan through

columns to detect the categorical column from all non-key columns and then match data values of those columns with the possible sub-classes of the matching class. Similarly challenges involving identification of pattern for automatic mapping systems have been discussed and solved in Chapter 4. Table 28 depicts the theoretical extent to which the state-of-the-art systems are designed to address the requirements for automatic mapping generation systems outlined in Table 7. Table 28 below depicts the theoretical extent to which Milan is designed to address the requirements.

It is argued that Milan shows improvement from the state the art in several requirements. It utilizes both the source relational database and target ontology. It also has a dedicated label matcher (4.3.1) which has customized variants based on specific relational-to-ontology patterns. Table 28 provides positive indications as to how Milan's capabilities can address structural heterogeneity, as it promises to address normalization challenges, class hierarchies and dependency conflicts. This is a significant design enhancement from the state-of-the-art systems. It is also designed to address semantic heterogeneity, arising due to impedance mismatch. Milan's support for OWL constructs also promises better performance at bridging the inter-model gap.

Table 28 shows to what extent Milan addresses the requirements for an automatic relational-to-ontology mapping system. Key aspects that differential Milan from existing systems includes support for 1:n, n:1 and m:n table-class correspondences, use of Hungarian algorithm for solving assignment problems in table-class correspondences, Tokenization in lexical matcher, customization for label matching between relational database and ontology. For example, exclusion of super class labels while performing 1:n table-class mapping detection which results in higher accuracy match results due to more distinctive results between two strings (refer to *filtered_label_matching* in Section 4.3.1).

6.2.3. RO3: Evaluate the prototype over multiple datasets and compare results with results from other state-of-the-art mapping generation systems

This research objective was fulfilled by a rigorous experimental evaluation of Milan across benchmark scenarios (datasets) that can closely mimic the challenges encountered by mapping generation in the real world, and finally comparing all the results based on the defined mapping quality metric (f-measure). Chapter 5 discusses the hypothesis, scenarios, quality metric and the methodology in detail. This is followed by the results, comparisons and finally an analysis of Milan's success and failure in the evaluation.

This evaluation was performed using a third party, widely accepted RODI benchmarking tool [5]. It computes overall mapping correspondence quality score using f-measure as well as scores for individual mapping challenge categories identified as Tag IDs in Table 29. For example, n-1 refers to detecting n:1 table-class correspondences. Four RODI input scenarios, each varying in size, complexity and domain. This diversity is reflected in Table 30. They were selected to offer realistic and varied mapping challenges which holistically test all the mapping patterns discussed in Table 7.

The result of the evaluation shows that Milan outperforms the seven other state-of-the-art systems for which comparable RODI benchmarking data is available. The average f-measure is 0.535 across four scenarios ranging between 0.30 to 0.86. It can be inferred that the quality of Milan's generation of mapping correspondence is 53.5% of the ideal. While it can only partially solve the relational-to-ontology mapping challenge, it performs better than the current state-of-the-art systems. It shows an improvement of 18.4% from the next best system on evaluations over the RODI benchmarking system [5].

Section 5.3 highlights the successes and failures of Milan in the evaluation. The analysis of Section 5.3 is drawn from the comparison of overall results discussed in Section 5.2.1, comparison on the basis of class, attributes and links in Section 5.2.2 and mapping correspondence quality for each mapping challenge

(Tag ID) across all the scenarios in Section 5.2.3. For example, while, Milan performed as well as the other top systems in detecting class-table correspondences, Milan's performed significantly better under "attrib" tag ID (37% improvement) and link tag ID (34% improvement) correspondences than the next best. The next example is where Milan fails to perform in the evaluation - Milan couldn't capture object property mappings when object properties in the ontology do not have explicit rdfs:range/domain constructs (i.e. for more complex OWL models).

6.3. Future Work

Milan has scope for future research, both in the design and the evaluation of Milan. This section identifies possible future work to improve the performance of Milan.

On the design front, the future work is directly impacted by the evaluation results of Table 41, which shows the f-scores across Tag IDs. Last column of Table 41 indicates the extent of the scope for improvement of Milan. Top Tag IDs in the scope for improvement are *"no-fk", "n-1" and "join-n"*.

These include, providing support for "*no-fk*" Tag ID (non-existing foreign key – primary key relationships) by inferring implicit foreign key to primary key relationships to address ST11 relational-to-ontology mapping challenge of Table 7. This would enable matching of object properties to non-existing foreign key-primary key relationship. For example, class *employee* and *department* are the domain and range respectively of object property *emp_depatment*. Similarly in the database, the corresponding tables are *emp* and *dept*. While column *emp_dep* of *emp* contains the primary key values of *dept*, the relational database meta-data does not contain the foreign key-primary key relationship. In such cases, Milan will be equipped to infer and finally match it to *emp_dep* object property.

Another possible extension to the Milan algorithm is enhancements in 1:n table-class mapping correspondence detection by extending the column-based table splitting pattern to account for columns containing Boolean values and foreign

keys. This has been discussed with examples, as failure at mapping challenges in Section 5.3.

The *"join-n"* Tag ID can be addressed by extending Milan's capability of using junction tables to detect n:m relations using a single junction tables. This would require Milan to detect multiple junction tables to establish relationship between two or more tables.

Another enhancement is adding further support for inferring more OWL 2 constructs to resolve semantic heterogeneity. For example, processing `owl:maxCardinality and owl:minCardinality`. The need for this is shown in from the effective use of rdfs:label, which itself improves the performance of Milan by 12% in a particular scenario. Refer to success at mappings sub-section of Section 5.3. Also, translating Milan's current output into robust mapping languages such as R2RML. This will allow end-to-end data integration from relational databases to ontologies.

On the evaluation front, future work includes, gathering evaluation scores (f-measure) across different Tag IDs for each system. Currently, we only had overall scores. This will allow greater granularity and objectivity while distinguishing where and how Milan is better than existing systems. Also, while the current RODI evaluation methodology is the most comprehensive benchmarking tool, its scenarios deal with real or realistic data only. The intention is to add scenarios which extreme challenges to stress test systems. For example, if the labels of matching table-class pair were to be ("*xyz*", "*abc*"), it is sure that string matching alone would fail to detect. It would be intriguing to know how systems including Milan perform in such scenarios.

6.4. Final Remarks

It is hoped by the author of this thesis that, this thesis has motivated the reader to appreciate the mapping generation research problem, its merits both in the context of technical complexity as well as the potential for impact on the application

domain. Automatic relational-to-ontology mapping systems are increasingly instrumental in undertaking the mammoth data integration tasks of modern digital enterprise, reducing human effort and dealing with ever evolving data both in size and structure. It is hoped that Milan, as presented in this thesis, has contributed to getting us closer to achieving complete automation in relational-to-ontology mapping systems. The author is optimistic about enhancements to Milan, and also development of new systems using re-usable components which were designed as part of this thesis.

The research masters was a great learning experience. I am proud to claim that at the beginning of this course, I was completely new to the field, and the only motivation towards this research was my interest in graphs. I believe my learning during this research enabled me to be more inquisitive and more rigorous. I am very excited to extent the current version of Milan and therefore hope to create a further impact in the scientific community. I sincerely believe that this thesis was an opportunity to learn the art of communication, not just academic but holistic.

References

[1] P. Shvaiko and J. Euzenat, “Ontology Matching: State of the Art and Future Challenges,” *IEEE Trans. Knowl. Data Eng.*, vol. 25, no. 1, pp. 158–176, Jan. 2013.

[2] J. Sequeda, F. Priyatna, and B. Villazón-Terrazas, “Relational Database to RDF Mapping Patterns,” in *Proceedings of the 3rd International Conference on Ontology Patterns - Volume 929*, Aachen, Germany, Germany, 2012, pp. 97–108.

[3] M. Hert, G. Reif, and H. C. Gall, “A Comparison of RDB-to-RDF Mapping Languages,” in *Proceedings of the 7th International Conference on Semantic Systems*, New York, NY, USA, 2011, pp. 25–32.

[4] “R2RML: RDB to RDF Mapping Language.” [Online]. Available: https://www.w3.org/TR/r2rml/. [Accessed: 27-May-2018].

[5] C. Pinkel *et al.*, “RODI: Benchmarking relational-to-ontology mapping generation quality,” *Semantic Web Journnal*, vol. 9, no. 1, pp. 25–52, 2018.

[6] J. D. Ullman, H. Garcia-Molina, and J. Widom, *Database Systems: The Complete Book*, 1st ed. Upper Saddle River, NJ, USA: Prentice Hall PTR, 2001.

[7] M. Rodríguez-Muro, R. Kontchakov, and M. Zakharyaschev, “Ontology-Based Data Access: Ontop of Databases,” in *The Semantic Web – ISWC 2013*, 2013, pp. 558–573.

[8] A. Poggi, D. Lembo, D. Calvanese, G. D. Giacomo, M. Lenzerini, and R. Rosati, “Linking Data to Ontologies,” in *Journal on Data Semantics X*, Springer, Berlin, Heidelberg, 2008, pp. 133–173.

[9] E. Jiménez-Ruiz *et al.*, “BootOX: Practical Mapping of RDBs to OWL 2,” in *The Semantic Web - ISWC 2015*, 2015, pp. 113–132.

[10] C. Pinkel *et al.*, *IncMap: A Journey towards Ontology-based Data Integration*. Gesellschaft für Informatik, Bonn, 2017.

[11] D. Calvanese *et al.*, “Ontop: Answering SPARQL queries over relational databases,” *Semantic Web*, vol. 8, no. 3, pp. 471–487, Jan. 2017.

[12] L. F. de Medeiros, F. Priyatna, and O. Corcho, “MIRROR: Automatic R2RML Mapping Generation from Relational Databases,” in *Engineering the Web in the Big Data Era*, 2015, pp. 326–343.

[13] H.-H. Do and E. Rahm, “COMA: A System for Flexible Combination of Schema Matching Approaches,” in *Proceedings of the 28th International Conference on Very Large Data Bases*, Hong Kong, China, 2002, pp. 610–621.

[14] C. Bizer, “D2RQ - treating non-RDF databases as virtual RDF graphs,” in *In Proceedings of the 3rd International Semantic Web Conference (ISWC2004*, 2004.

[15] Á. Sicilia and G. Nemirovski, “AutoMap4OBDA: Automated Generation of R2RML Mappings for OBDA,” in *Knowledge Engineering and Knowledge Management*, 2016, pp. 577–592.

[16] C. Batini, M. Lenzerini, and S. B. Navathe, “A comparative analysis of methodologies for database schema integration,” *ACM Comput. Surv. CSUR*, vol. 18, no. 4, pp. 323–364, 1986.
[17] M. Poess, T. Rabl, H.-A. Jacobsen, and B. Caufield, “TPC-DI: The First Industry Benchmark for Data Integration,” *Proc VLDB Endow*, vol. 7, no. 13, pp. 1367–1378, Aug. 2014.
[18] D. Tarasowa, C. Lange, and S. Auer, “Measuring the quality of relational-to-RDF mappings,” in *International Conference on Knowledge Engineering and the Semantic Web*, 2015, pp. 210–224.
[19] L. F. de Medeiros, F. Priyatna, and O. Corcho, “MIRROR: Automatic R2RML Mapping Generation from Relational Databases,” in *Engineering the Web in the Big Data Era*, 2015, pp. 326–343.
[20] C. Goutte and E. Gaussier, “A Probabilistic Interpretation of Precision, Recall and F-Score, with Implication for Evaluation,” in *Advances in Information Retrieval*, 2005, pp. 345–359.
[21] E. F. Codd, “A Relational Model of Data for Large Shared Data Banks,” *Commun ACM*, vol. 13, no. 6, pp. 377–387, Jun. 1970.
[22] “MySQL :: MySQL 8.0 Reference Manual :: 24 INFORMATION_SCHEMA Tables,” *MySQL*. [Online]. Available: https://dev.mysql.com/doc/refman/8.0/en/information-schema.html. [Accessed: 16-May-2018].
[23] T. Gruber, “Ontology,” in *Encyclopedia of Database Systems*, Springer, Boston, MA, 2009, pp. 1963–1965.
[24] C. Batini, M. Lenzerini, and S. B. Navathe, “A Comparative Analysis of Methodologies for Database Schema Integration,” *ACM Comput Surv*, vol. 18, no. 4, pp. 323–364, Dec. 1986.
[25] T. Berners-lee, J. Hendler, and O. Lassila, “The Semantic Web,” *Sci. Am.*, vol. 284, no. 5, pp. 34–43, 2001.
[26] “A Primer on Machine Readability for Online Documents and Data,” *Data.gov*, 24-Sep-2012. [Online]. Available: https://www.data.gov/developers/blog/primer-machine-readability-online-documents-and-data. [Accessed: 17-May-2018].
[27] D. Brickley and R. V. Guha, “RDF Schema 1.1,” *W3C Recommendation*. [Online]. Available: https://www.w3.org/TR/rdf-schema/. [Accessed: 24-Apr-2018].
[28] Prud’hommeaux, Eric and Seaborne, Andy, “SPARQL Query Language for RDF.” [Online]. Available: https://www.w3.org/TR/rdf-sparql-query/. [Accessed: 24-Apr-2018].
[29] “RDF 1.1 Primer.” [Online]. Available: https://www.w3.org/TR/rdf11-primer/. [Accessed: 27-Apr-2018].
[30] Patrick J. Hayes and Peter F. Patel-Schneider, “RDF 1.1 Semantics.” [Online]. Available: https://www.w3.org/TR/rdf11-mt/.
[31] E. Miller, “An Introduction to the Resource Description Framework,” Corporation for National Research Initiatives, 1998.

[32] “Ontologies - W3C.” [Online]. Available: https://www.w3.org/standards/semanticweb/ontology. [Accessed: 24-Apr-2018].
[33] Deborah L. McGuinness and Frank van Harmelen, “OWL Web Ontology Language Overview.” [Online]. Available: https://www.w3.org/TR/owl-features/. [Accessed: 26-Apr-2018].
[34] Pascal Hitzler, Markus Krötzsch, and Bijan Parsia, “OWL 2 Web Ontology Language Primer.” [Online]. Available: https://www.w3.org/TR/rdf-schema/.
[35] “RDF Schema 1.1.” [Online]. Available: https://www.w3.org/TR/rdf-schema/. [Accessed: 26-Apr-2018].
[36] “RDF Data Range.” [Online]. Available: https://www.w3.org/TR/owl-ref/#DataRange. [Accessed: 22-Apr-2018].
[37] “SPARQL 1.1 Query Language.” [Online]. Available: https://www.w3.org/TR/sparql11-query/. [Accessed: 03-Apr-2018].
[38] “SPARQL endpoint.” [Online]. Available: http://semanticweb.org/wiki/SPARQL_endpoint.html.
[39] M. Šeleng, M. Laclavík, Z. Balogh, and L. Hluchỳ, “RDB2Onto: Approach for creating semantic metadata from relational database data.”
[40] E. Kharlamov *et al.*, “Ontology-Based Integration of Streaming and Static Relational Data with Optique,” in *Proceedings of the 2016 International Conference on Management of Data*, New York, NY, USA, 2016, pp. 2109–2112.
[41] H. A. Santoso, S.-C. Haw, and Ziyad. T. Abdul-Mehdi, “Ontology extraction from relational database: Concept hierarchy as background knowledge,” *Knowl.-Based Syst.*, vol. 24, no. 3, pp. 457–464, Apr. 2011.
[42] O. Corby, C. Faron-Zucker, and F. Gandon, “A Generic RDF Transformation Software and Its Application to an Online Translation Service for Common Languages of Linked Data,” in *The Semantic Web - ISWC 2015*, 2015, pp. 150–165.
[43] J. F. Sequeda, M. Arenas, and D. P. Miranker, “On Directly Mapping Relational Databases to RDF and OWL (Extended Version),” *ArXiv12023667 Cs*, Feb. 2012.
[44] D.-E. Spanos, P. Stavrou, and N. Mitrou, “Bringing relational databases into the Semantic Web: A survey,” *Semantic Web*, vol. 3, no. 2, pp. 169–209, Jan. 2012.
[45] F. Freitas, S. Schulz, and E. Moraes, “Moraes E: Survey of current terminologies and ontologies in biology and medicine,” in *RECIIS - Electronic Journal in Communication, Information and Innovation in Health*, pp. 7–18.
[46] E. Jiménez-Ruiz and B. C. Grau, “Logmap: Logic-based and scalable ontology matching,” in *International Semantic Web Conference*, 2011, pp. 273–288.
[47] A. Solimando, E. Jiménez-Ruiz, and G. Guerrini, “Detecting and Correcting Conservativity Principle Violations in Ontology-to-Ontology Mappings,” in *The Semantic Web – ISWC 2014*, 2014, pp. 1–16.

[48] W. F. Dowling and J. H. Gallier, “Linear-time algorithms for testing the satisfiability of propositional horn formulae,” *J. Log. Program.*, vol. 1, no. 3, pp. 267–284, Oct. 1984.
[49] D. Aumueller, H.-H. Do, S. Massmann, and E. Rahm, “Schema and Ontology Matching with COMA++,” in *Proceedings of the 2005 ACM SIGMOD International Conference on Management of Data*, New York, NY, USA, 2005, pp. 906–908.
[50] G. Stoilos, G. Stamou, and S. Kollias, “A String Metric for Ontology Alignment,” in *The Semantic Web – ISWC 2005*, Springer, Berlin, Heidelberg, 2005, pp. 624–637.
[51] E. Kharlamov *et al.*, “How Semantic Technologies Can Enhance Data Access at Siemens Energy,” in *The Semantic Web – ISWC 2014*, Springer, Cham, 2014, pp. 601–619.
[52] E. Kharlamov *et al.*, “Ontology Based Access to Exploration Data at Statoil,” in *Proceedings, Part II, of the 14th International Semantic Web Conference on The Semantic Web - ISWC 2015 - Volume 9367*, New York, NY, USA, 2015, pp. 93–112.
[53] S. Melnik, H. Garcia-Molina, and E. Rahm, “Similarity flooding: a versatile graph matching algorithm and its application to schema matching,” in *Proceedings 18th International Conference on Data Engineering*, 2002, pp. 117–128.
[54] C. Pinkel, C. Binnig, E. Kharlamov, and P. Haase, “Pay-as-you-go Matching of Relational Schemata to OWL Ontologies with IncMap,” in *Proceedings of the 12th International Semantic Web Conference (Posters & Demonstrations Track) - Volume 1035*, Aachen, Germany, Germany, 2013, pp. 225–228.
[55] M. Cheatham and P. Hitzler, “String Similarity Metrics for Ontology Alignment,” in *The Semantic Web – ISWC 2013*, 2013, pp. 294–309.
[56] E. W. Dijkstra, “A Note on Two Problems in Connexion with Graphs,” *Numer Math*, vol. 1, no. 1, pp. 269–271, Dec. 1959.
[57] F. Cerbah, “Mining the Content of Relational Databases to Learn Ontologies with Deeper Taxonomies,” in *Proceedings of the 2008 IEEE/WIC/ACM International Conference on Web Intelligence and Intelligent Agent Technology - Volume 01*, Washington, DC, USA, 2008, pp. 553–557.
[58] D. Calvanese *et al.*, “Ontop: Answering SPARQL queries over relational databases,” *Semantic Web*, vol. 8, no. 3, pp. 471–487, 2017.
[59] V. I. Levenshtein, “Binary Codes Capable of Correcting Deletions, Insertions and Reversals,” *Sov. Phys. Dokl.*, vol. 10, p. 707, Feb. 1966.
[60] O. Corcho, M. Poveda-Villalón, and A. Gómez-Pérez, “Ontology engineering in the era of linked data,” *Bull. Assoc. Inf. Sci. Technol.*, vol. 41, no. 4, pp. 13–17, Apr. 2015.
[61] A. Isaac, L. van der Meij, S. Schlobach, and S. Wang, “An Empirical Study of Instance-Based Ontology Matching,” in *The Semantic Web*, Springer, Berlin, Heidelberg, 2007, pp. 253–266.
[62] H. W. Kuhn, “The Hungarian method for the assignment problem,” *Nav. Res. Logist. Q.*, vol. 2, no. 1–2, pp. 83–97, 1955.

[63] “Mapping SQL datatypes to XML Schema datatypes.” [Online]. Available: https://www.w3.org/2001/sw/rdb2rdf/wiki/Mapping_SQL_datatypes_to_XML_Schema_datatypes. [Accessed: 24-May-2018].
[64] “Ontology Alignment Evaluation Initiative::Home.” [Online]. Available: http://oaei.ontologymatching.org/. [Accessed: 16-Aug-2018].
[65] Z. Dragisic *et al.*, “Results of the Ontology Alignment Evaluation Initiative 2014,” in *Proceedings of the 9th International Conference on Ontology Matching - Volume 1317*, Aachen, Germany, Germany, 2014, pp. 61–104.
[66] M. Cheatham *et al.*, “Results of the Ontology Alignment Evaluation Initiative 2015,” in *10th ISWC workshop on ontology matching (OM)*, Bethlehem, United States, 2015, pp. 60–115.