1-1-2015

Semantic Web Based Relational Database Access With Conflict Resolution

Fayez Khazalah
Wayne State University,

Recommended Citation

Khazalah, Fayez, "Semantic Web Based Relational Database Access With Conflict Resolution" (2015). *Wayne State University Dissertations.* Paper 1315.

SEMANTIC WEB BASED RELATIONAL DATABASE ACCESS WITH CONFLICT RESOLUTION

by

FAYEZ S. KHAZALAH

DISSERTATION

Submitted to the Graduate School

of Wayne State University,

Detroit, Michigan

in partial fulfillment of the

requirements for the degree of

DOCTOR OF PHILOSOPHY

2015

MAJOR: COMPUTER SCIENCE

Approved by:

Advisor Date

DEDICATION

To

my MOTHER and FATHER

my Wife

my Kids, Yamen, Bushra, and Saleem

my Brothers, Sisters, and their families

TABLE OF CONTENTS

LIST OF FIGURES

LIST OF TABLES

PART I: THESIS INTRODUCTION

CHAPTER 1: Introduction and Problem Statement

The vision of the Semantic Web was introduced in the early part of this century to convert the existing World Wide Web from the *Web of Documents* to the *Web of Data*. The latter defines a medium where data is structured and classified into taxonomies of concepts, attributes, and the relations between those concepts and attributes (i.e., ontological) (Doan et al., 2002). Since the Semantic Web's inception (and long before that), relational database systems have been used for storing the resulting data; due mainly to the maturity, performance and efficiency of the relational database model (Beckett and Grant, 2003). This has led to the development of dedicated database systems (a.k.a. triple stores) for storing the Semantic data (del Mar Roldan-Garcia and Aldana-Montes, 2006). However, in recent years the research focus has shifted towards supporting the access to the already existing and massive data (usually stored in relational databases). The motivation behind this direction is to bring the relational database communities into the Semantic Web world. Statistics show that more than two-thirds of the data on the Internet is stored in structured or relational databases (Chang et al., 2004). The data items are mostly hidden from search engines and Web crawlers, and can only be accessed through dynamic Web pages, generated per user queries (using interfaces by data owners, or invoking Web services) (Geller et al., 2008). Since semantic data is scarce at best, the Semantic Web community will not be able to achieve higher growth rates and productivity without the involvement of relational databases.

From the viewpoint of the relational database community, a complete transition from the relational model into the Semantic Web is a complex and difficult decision at best. Therefore, we believe that encouraging the database community to adopt the Semantic Web technology is more effective if it is accomplished in a gradual and smooth manner. In recent years, technologies have been proposed to allow the extraction of an ontology from a relational source, defining mappings between a relational source and an existing ontology, and running user queries over the defined ontology and getting back answers from the relational source that is mapped to the ontology. These efforts can be divided into two distinct areas (Spanos et al., 2012): The first area aims at extracting an ontology from an existing relational schema, while the other area aims at finding mappings between an existing relational source and an existing ontology. In this thesis, we focus on the former.

Developing a new ontology from scratch is usually very difficult and costly (both in monetary and chronological terms), and requires expert opinions from the relevant domain of knowledge. Thus, ontology learning techniques have been proposed. These techniques have been used later in ontology-based data access (OBDA) (Sequeda et al., 2009). OBDA assumes that there is a mediating ontology between end users and a data source, such that each entity in the relational schema is mapped to an equivalent entity in the ontology, in such a way that end users are not required to know about the underlying source schema structure (Poggi et al., 2008). Recently, the *Ontop*[1] framework (Rodriguez-Muro and Calvanese, 2012) has positioned itself as one of the competitive approachs for OBDA (after the success of the well-known D2RQ framework). For instance, in a recent evaluation study (Rodrıguez-Muro et al., 2013) it is shown that the *Ontop* framework is efficient, and achieves good performance. *Ontop* deploys query rewriting techniques with Semantic Query Optimization in an efficient manner. Thus, the queries execute faster. Moreover, redundant joins are eliminated in the optimization process, which is beneficial when SQL queries are written by inexpert users. (Rodriguez-Muro et al., 2013) also shows that the performance of the SQL queries that are generated by *Ontop* are superior, compared to

[1]http://ontop.inf.unibz.it

other systems that translate SPARQL queries into SQL (e.g., D2RQ and Virtuoso RDF Views), and other well known triple stores (e.g., OWLIM, Stardog, and Virtuoso). We posit that the continuous development of the *Ontop* framework by community, its open-source, in addition to providing a tool to translate R2RML mappings (Cyganiak et al., 2012) into *Ontop* mappings and vice versa, will increase the popularity of the *Ontop* framework, making it potentially one of the leading solutions for ontology-based data access.

Manually extracting *Ontop* mapping rules and OWL ontology from a relational schema is a very complex, time consuming process, and it needs expert people to complete the job. Therefore, automated approaches for the extraction process are needed to facilitate, speed up this process, and minimize the need for expert people. One of the primary uses for Ontology Based Data Access approaches (e.g., Ontop) is for integration of distributed relational databases that either have heterogeneous schemata , heterogeneous data instances, or both. An example on such an environment where we can apply Semantic Web based data integration is in collaborative sharing systems, such as CDSS. However, we need to address the problem of conflict resolution in CDSS if we want to use Ontology based data access approaches for the collaborative sharing and integration. This is what we are discussing next.

In collaborative data sharing systems, groups of users usually work on disparate schemas and database instances, and agree to share the related data among them (periodically). Each group can extend, curate, and revise its own database instance in a disconnected mode. At some point later, the group can publish its updates to other groups and get updates of other ones (if any). The reconciliation operation in the CDSS engine is responsible for propagating updates and handling any data disagreements between the different groups. If a conflict is found, any involved updates are rejected temporally and marked as deferred. Deferred updates are not accepted by the reconciliation operation until a user resolves the conflict manually.

The administrator of each peer in a CDSS is usually responsible for declaring and managing trust policies. While the administrator can be expected to define trust policies for a small number of participant peers, the same is not true for a large number of participants. In addition, assuming that a community of hundreds or thousands of members can authorize a user or a group of users to define trust policies for their community may not be plausible. Moreover, a CDSS does provide a semi-automatic conflict resolution approach by accepting the highest-priority conflicting updates, but it leaves for individual users the responsibility of resolving conflicts for the updates that are deferred. However, the assumption that individual users can decide how to resolve conflicting updates is not strong, as users of the community may have different beliefs and may agree or disagree with each other about which conflicting updates to accept and why (i.e., on which bases). Therefore, the challenge lies in providing a conflict resolution framework that requires minimal or no human intervention.

1.1 Research Contributions

The main contributions of this thesis can be summarized in the following points:

- We propose a Semantic Web based approach for accessing relational data. In the proposed approach, Ontop mappings and OWL ontology are extracted from a given relational source. At the beginning, we classify a relational source entities into different relation types (i.e., independent, dependent, recursive, binary, n-ary, inheritance, partitioning, etc.). We then define an Ontop mapping rule's template for each entity type. An algorithm is used to generate each of these templates. The extractor module uses the templates to extract Ontop mapping rules for a relational source. In addition, it extracts an OWL ontology that is equivalent to the relational source's schema. Finally, a SPARQL endpoint that is built on top of Ontop's Quest reasoner is used to access relational data through the extracted OWL ontology and the extracted Ontop mappings rules. We summarize this contribution below:

  – The proposal of an automatic *Ontop* mappings rules and ontology extraction from a relational schema.
  – A prototype system implementation of the proposed approach. We have implemented the proposed approach in Java. The end-users can access any remote relational data through a JDBC connection. After establishing the connection, end-users can extract the *Ontop* mapping rules and OWL ontology from the underlying data source, pose SPARQL queries over the extracted ontology (to access the relational data) and get the results back. In addition, end-users can alter both the extracted rules and the ontology according to their needs.
  – A set of experiments have been conducted to evaluate the proposed approach. We have evaluated the proposed approach using different freely available relational databases. The evaluation process is composed of two steps: In the first step, the ontology and mapping rules are extracted. In the second step, SPARQL queries (over the extracted ontologies) are used to access the relational data sources.

- We define a novel conflict resolution approach that extends the CDSS to automate the resolution of conflicts in the deferred set of a CDSS's reconciling peer. We define a distributed trust mechanism to compute the weight for each conflicting update.

1.2 Dissertation Organization

The remaining of the dissertation is organized as follows. Chapter 2 is a background on Semantic Web and ontology-based data access. We present our proposed approach for extracting mapping rules and OWL ontology from a relational schema in Chapter 3. Chapter 4 gives an overview for collaborative sharing systems and conflict resolutions. We finally introduce our proposed approach for resolving conflicts automatically in Collaborative Data Sharing System (CDSS) in Chapter 5.

PART II: SEMANTIC WEB BASED RELATIONAL DATABASE ACCESS WITH ONTOLOGY EXTRACTION AND MAPPINGS

CHAPTER 2: Ontop and the Semantic Web

As mentioned in the introduction chapter (1), *Ontop* is one of the widely used framework for OBDA. However, mappings extraction is not the core focus of the *Ontop* research group (initially assuming that the *Ontop* mappings can be written manually), the recent release added an automatic extraction option for *Ontop* mappings and ontology from an existing data source in the *OntoPro* plugin for Protegé (Rodriguez-Muro et al., 2008). However, this tool follows a basic approach for automating the process of extraction. Namely, it simply extracts an ontology that is an exact copy of the relational schema and does not consider the relationships between relational entities. This complicates the process of mapping the extracted ontology with domain ontologies that have a rich structure than that of a relational schema. Some other shortcomings and incompatibilities in the existing extracted *Ontop* mappings are defined in the following:

- It does not recognize a binary relation. Instead, it extracts incorrect *Ontop* mappings for representing binary relations.

- It represents the n-ary relation as n separate relations between the n relations that are composing the n-ary relation. Thus, it may require up to n separate SQL joins to retrieve the data that represents the n-ary join relation (which is clearly inefficient).

- It does not recognize a recursive reference. Thus, it fails to extract *Ontop* mapping rules for representing recursive references.

- It does not recognize a fragmented table that is represented using more than one table. When extracting ontology concepts from the tables that represent the fragmented table, only one concept should be extracted for all fragmented tables. In other words, one OWL (Motik et al., 2012) concept is generated, and all attributes in the fragments are represented as OWL properties that have the extracted concept as their domain.

- It does not recognize the inheritance relationship between tables (i.e., IS-A relationship). Finding the inheritance relationship is sometimes ambiguous with the fragmented table, because in both fragments and inheritance, all tables have the same primary key. However, they can be distinguished by analyzing the data instances using data mining techniques; which is only available when there is ample data.
- It ignores other database constraints (e.g., check, null, not null, etc.) and multi-valued columns (e.g., enum etc.).

To overcome the above mentioned limitations, we propose an approach for automating the process of *Ontop* mappings extraction from an existing database schema. The proposed approach considers the different relationships (binary vs. n-ary etc.) between the entities of the schema. It extracts *Ontop* mapping rules based on the type of a database table, where the possible types are independent, dependent, recursive, binary, and n-ary relation. An Independent table is a primary or a master table that is not dependent on other tables. In other words, it does not have any foreign key that references other tables. Extracting the *Ontop* mappings for representing the Independent table rule is a straightforward. On the contrary, a Dependent table has at least one attribute that is a foreign key referencing another table. A Recursive table can thus be classified under the dependent table. However, the *Ontop* mapping rule for representing the recursive relation has a different format than that for representing the non-self referencing tables. A Binary Join table represents the relation between two tables. This relation is a many-to-many, that can be divided into two one-to-many relations, one from the first table to the second and vice versa. When extracting *Ontop* mappings for a Binary Join, it very important to consider whether the Binary Join table has some other non-key attributes or not. In the case that it does not have any non-key attributes, we should not extract an equivalent class for this table. It is enough just to extract two object properties for representing each one-to-many relation that is composing the binary join. However, if there are some non-key attributes in the binary join table, we deal with it the same way as in the n-ary join table. The n-ary join table represents the relation between two or more tables. We cannot represent the n-ary relation directly in the OWL ontology. Instead, we extract an object property that represents this relation. The domain of the extracted object property is the equivalent class that is extracted from the n-ary join table. It also has (as its range) the group of extracted classes that are equivalent to the underlying n tables that are composing the n-ary join table. We thus do not break the n-ary join relation into multiple binary relations as it is done in other approaches.

2.1 Semantic Web Overview

The majority of the current Web's content can only be translated by humans. The Semantic Web is a new paradigm that establishes the foundation for the next generation of the visioned web (aka Web 3). The goal of Semantic Web is to improve the current state of the Web by making the information accessible and processed via machines. We discuss below the most popular technologies and tools that exist nowadays for representing, defining, and deploying Semantic Web applications.

2.1.1 Ontologies

The history of the term *ontology* returns back a long time ago to a subfield of philosophy that was dedicated for studying the nature of existence (Antoniou, 2008). It tries to identify the existing things using general terms and discusses they that they can be described. Recently, the term *ontology* has been used in the computer science field in different meaning. An ontology is defined as "an explicit and formal specification of a conceptualization" (Antoniou, 2008). An ontology is usually a domain based. Even more, the same domain can be represented by different ontologies. An ontology is composed of concepts and properties that describe the relationships between these concepts. A concept is denoted by a class and a relationship is denoted by a property. In addition, an ontology supports hierarchical structures among classes and properties. In other words, a one class can be a sub-class from another class and a property can be a sub-property from another property. Nowadays, ontologies can be represented over the Web by different ontology languages. The most popular languages are RDF, RDF Schema, and OWL. We next provide brief details of them.

2.1.2 RDF

The Resource Description Framework (RDF) (Wood et al., 2014) is a W3C standard language for exchanging data over the Web. The core of the RDF is simply describing Web resources by using the concept of a triple that consists of three parts: subject, predicate and object. A triple is a statement that describes a particular property (stated in the predicate part) of a Web resource (stated in the subject part) by the value in the object part. It asserts that there is a relationship between resources (i.e., subject and object of the statement) that is represented by this particular predicate. A group of RDF triples forms an RDF graph, where the nodes are the subject or object of triples and the directed edge between any two nodes represents the relationship between them. The resource is simply anything in the world (i.e., physical things, documents, abstract concepts, numbers, strings, etc.). Every resource is identified by unique Internationalized Resource Identifier (IRI). The subject and predicate of a triple should be always an IRI resource, whereas the object of a triple can be either an IRI resource or a literal. A literal is a constant value of a specific data type (i.e., number, string, date, etc.) that restricts the possible values that can be assigned to this literal.

2.1.3 RDFS

The RDF Schema (RDFS) (Guha and Brickley, 2014) is a data-modeling vocabulary that semantically extends the basic RDF vocabulary. It provides new mechanisms to describe resources and their inter-relations. For example, it adds the terms rdfs:class to indicate the type of a resource, rdfs:domain and rdfs:range to determine the possible domains and ranges of a particular resource. RDFS is somehow similar to object-oriented paradigms. However, it differs in the way classes and properties are described. In object-oriented models, a class is defined by the properties that its instance may have. In the contrary, the properties in RDFS are described by the possible classes of resources that they may be applied to.

In RDFS, a resource is represented by a class as in object-oriented approach. A resource is said to be of a class type using RDFS class rdfs:Class. Each class is identified by an IRI and its features are described by RDF properties. The set of RDF triples that describe a class are called the instances of the class. An instance is stated to be a member of a class using RDF property rdf:type. Classes can also be represented hierarchically for supporting the inheritance by using RDFS property rdfs:subClassOf. For example, the triple ”A rdfs:subClassOf B”, says that the class A is a subclass of a class C. rdfs:Literal represents the class of literal values (i.e., numbers, strings, etc.).

2.1.4 OWL

OWL is an ontology language with a richer semantic for describing classes and their properties (Guha and Brickley, 2014). For example, it can describe equality or disjoint relations between classes, symmetric relations, cardinality, richer typing of properties, enumerated classes, etc.

2.1.5 SPARQL Query Language

SPARQL (Prud'hommeaux and Seaborne, 2008) is query language for RDF data. It is simply based on matching RDF graph patterns with the RDF data graph. An RDF graph pattern is a set of one or more triple patterns. A triple pattern is simply an RDF triple with variables represented by wild cards instead of using the RDF terms for subject, predicate, or object of an RDF triple. Given a SPARQL query pattern, the result of the query is computed by matching SPARQL query's RDF triple patterns with an RDF triple store (i.e., RDF graph). The RDF triples that are matched with the wild card variables of the query triple patterns are returned as results of the query.

2.2 Ontology Based Data Access (OBDA)

One of the primary benefits of linking relational databases with the Semantic Web is to extract Semantic data out of already existing data that resides inside relational databases. It is widely believed that a primary obstacle in the broader realization of the Semantic Web is the scarce number of tools, applications, and unavailability of adequate Semantic data (Konstantinou et al., 2010). In the early days of Semantic Web research, the focus was to transform all relational data into Semantic Web data (i.e., ETL process). The ETL process (Rodriguez-Muro et al., 2012) is composed of three steps. The first step is to Extract the data from the relational source. The second step is to Transform the extracted data into RDF triples or OWL instances in the target ontology. The last step is to Load those data into a SPARQL endpoint or an OWL reasoner. However, this approach has serious shortcomings. It duplicates the data storage, and does not guarantee that the current semantic data that is exported from a relational source is up-to-date, thus a re-export process for refreshing data is required, which is an impractical solution at best. In addition, the high cost of duplicating storage, especially when the size of the data is huge, makes it an undesirable solution.

OBDA approach has thence emerged as an alternative for transforming relational data into semantic terms that facilitates direct data access and does not require data transformation or multiple storage. The focus is therefore to provide tools for end-users to access data sources through a high-level conceptual view, that is presented using ontologies (Calvanese et al., 2007). It assumes the availability of an ontology that acts as an intermediate layer between the end-users, and the underlying data source (Spanos et al., 2012). However, the end-users are assumed not to be aware about the underlying database schema, structure of entities, and storage details (Poggi et al., 2008).

Many systems that provide direct access to relational data using ontology-based data access and SPARQL queries have been introduced in the literature (D2RQ server (Bizer and Seaborne, 2004), Virtuoso RDF (Erling and Mikhailov, 2010), Triplify (Auer et al., 2009), to name a few). However, these systems have some, but serious drawbacks (e.g., lack of the semantics support and poor query performance (Rodriguez-Muro et al., 2012)). The OBDA *Ontop* framework (Rodriguez-Muro and Calvanese, 2012) has been proposed to tackle the shortcomings of the existing approaches.

Ontop is an OBDA framework that supports on-the-fly SPARQL queries over RDBs through OWL and RDFS ontologies. *Ontop* is composed of two components: Quest and ontoPro. *Ontop Quest* (Rodriguez-Muro and Calvanese, 2012)(Rodriguez-Muro et al., 2012) is a Semantic Web inference system and SPARQL engine that comes with *Ontop*. In contrast with conventional RDF triple store that transform relational data into RDF triple before querying it, the *Quest* engine accesses the relational data and reasoning over it directly and on the fly, without transforming it into OWL assertions or RDF triples. Thus, it eliminates the performance issues related to memory limitations over large data. This mode of access is called *virtual ABox* mode. The *Quest* supports access to RDBs by using mapping rules (written in a mapping language) to translate SPARQL queries into SQL queries. It also deploys the query rewriting techniques efficiently and utilizes the high performance, scalability, and the maturity of the underlying RDBMS for executing and answering SPARQL queries. *Quest* uses a powerful mapping language introduced in (Poggi et al., 2008) for writing the *Ontop* mapping rules. A mapping rule is composed of two parts: a source part that is simply an SQL query, and a target part represents an ABox assertion template that is mapped with the source query. The template is simply a set of RDF triples written in Turtle format. The columns of the SQL query are mapped to the subject and object of the target's template triples, and the values of columns in the retrieved result are used to generate the virtual ABox assertions. *OntoPro* is a plugin that can be integrated into Protegé. It provides the required tools to connect to relational databases using JDBC, defines mappings between an

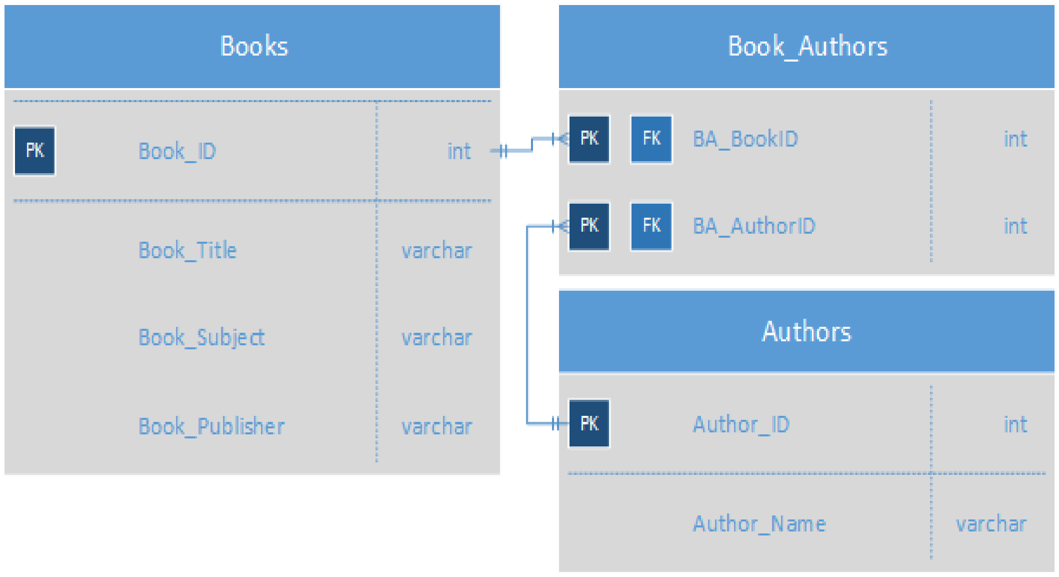


Figure 2.1: BookStore Schema.

active ontology and the database, and provides support using Quest reasoner to query relational data directly from Protege.

2.3 Motivating Scenario

In this section, we illustrate *Ontop* framework through an example. Assume we have a relational database schema (*BookStore*) that is shown in Figure 2.1. It stores and manages data about books and the authors of books. Assume we also have an OWL ontology (*OntoBookStore*) as shown in Figure 2.2 that is similar or equivalent to (*BookStore*), in such that both are representing the same domain of knowledge.

Through using *Ontop* framework, we can query the relational schema. *Ontop*'s mapping language provides the support for defining and managing mapping rules between the ontology and its equivalent or similar relational schema for the purpose of accessing the relational data. An *Ontop*'s mapping rule is simply an axiom that

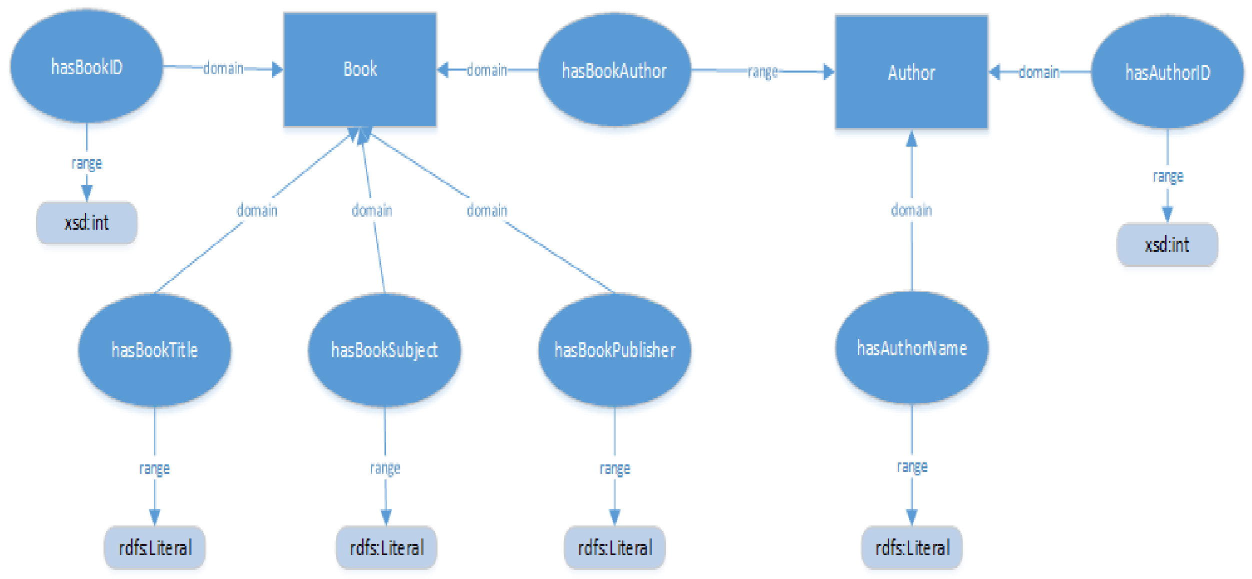


Figure 2.2: *OntoBookStore*, an equivalent ontology for *BookStore* schema.

relates relational entities and attributes to the correspondent concepts and properties of a similar or equivalent ontology.

We want, in this example, to declare the minimal number of *Ontop* mapping rules that are efficient and comprehensive for the purpose of querying the underlying relational data through SPARQL queries. We take a relational schema and an ontology as inputs and manually find a set of mapping rules that relates the relational schema's tables and fields to the classes and attributes of the ontology. The mechanism that we follow to declare the set of mappings is as follows:

- Find the mapping correspondences between entities/fields of the relational schema and concepts/properties of the equivalent ontology. For example, *Books* table in Figure 2.1 is paired with *Book* class in Figure 2.2.

- For each table/class pair found in above (i.e., *Books*/*Book* and *Authors*/*Author*), define a mapping rule that connects the table to its correspondent class from the ontology. The body of the mapping rule is composed of two parts: target and source. The target part is an ABox assertion template that maps the elements of the OWL ontology with their correspondent elements in the relational schema. It can also have one or more triples. The source part is simply an SQL query that represents all the database entities that are part of the mappings performed in the target part.

- Construct the subject template that represents the unique id of the virtual instance of the class that is derived from the related table. This subject template will plays as the subject for all the triples that belong to the mapping rule. The subject template is defined using the following format:

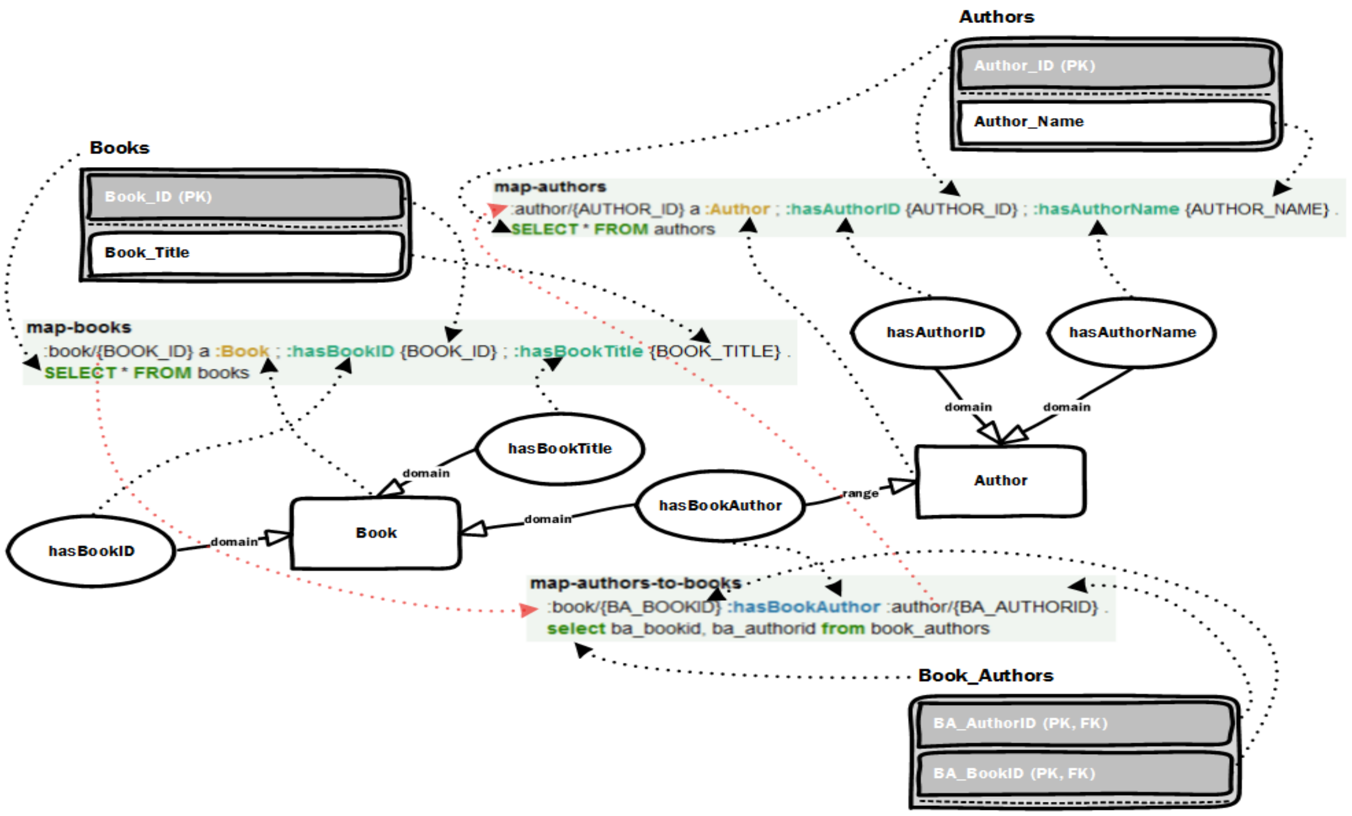
  $:< Class\_Name > /\{< Table\_Primary\_Key >\}$ where $< Table\_Primary\_Key >$ is the primary key of the relational table that is equivalent to the ontology class.

- Add the following triples to the target part of each rule:
  - a class triple that maps the given table with the equivalent class from the ontology.
  - a data property triple for each field in the table that maps the field in the table with the correspondent data attribute in the ontology.
  - an object property triple for each foreign key field in the table. It maps the foreign key and its referenced primary key in the other table with the correspondent relation in the ontology. The relation is simply an object property that has the class that is a correspondent to the foreign key's table as its domain and the class that is a correspondent to the primary key's table as its range.

- Write the appropriate SQL query and add it to the source part of the mapping rule.

- The remaining table (*Book_Authors*) does not have an equivalent class. However, this table is simply a binary-join relation on *Books* and *Authors* tables. If we look at the ontology, we will find that the object property *hasBookAuthor* is equivalent to the table *Book_Authors* as it represents the relation between *Book* and *Author* classes. Thus, to represent this binary relation we add a mapping rule with only one triple in the target part. The subject of the triple is a subject template for class *Book* with the part of the composite primary key (for table *Book_Authors*) that references table *Books* and the object is a subject triple for class *Author* with the second part of the composite primary key that references table *Authors*.

The extracted mapping rules are shown in Table 2.1. Figure 2.3 also shows a pictorial representation for the mapping rules and their associations with the relational schema and the OWL ontology, where the black dotted arrows represent the mappings between the elements of (the relational schema and ontology) and the mapping rules, and the red dotted arrows represent the relations among mapping rules. The motivating example shows that extraction of ontology and *Ontop* mapping rules from a relational schema is tedious and needs much time in addition to expert people. Therefore, there is a need to automate the extraction process.



```
map-authors
:author/{AUTHOR_ID} a :Author ; :hasAuthorID {AUTHOR_ID} ; :hasAuthorName {AUTHOR_NAME} .
SELECT * FROM authors

map-books
:book/{BOOK_ID} a :Book ; :hasBookID {BOOK_ID} ; :hasBookTitle {BOOK_TITLE} .
SELECT * FROM books

map-authors-to-books
:book/{BA_BOOKID} :hasBookAuthor :author/{BA_AUTHORID} .
select ba_bookid, ba_authorid from book_authors
```

Ontop mapping between BookStore database schema and its equivalent OWL ontology.

Figure 2.3: A pictorial mapping between *BookStore* schema, *OntoBookStore* ontology, and the mapping rules.

Table 2.1: Mapping rules between *OntoBookStore* ontology and *BookStore* relational schema.

1. map-books:

   TARGET: : *book*/{*BOOK_ID*} *a* : *Book*; **:hasBookID** {*BOOK_ID*}; **:hasBookTitle** {*BOOK_TITLE*};

   SOURCE: **SELECT** $*$ **FROM** *books*

2. map-authors:

   TARGET: : *author*/{*AUTHOR_ID*} *a* : *Author*; **:hasAuthorID** {*AUTHOR_ID*}; **:hasAuthorName** {*AUTHOR_NAME*};

   SOURCE: **SELECT** $*$ **FROM** *authors*

3. map-authors-to-books:

   TARGET: : *book*/{*BA_BOOKID*} : *hasBookAuthor* : *author*/{*BA_AUTHORID*};

   SOURCE: **SELECT** *ba_bookid*, *ba_authorid* **FROM** *book_authors*

CHAPTER 3: Ontop Mapping Rules and OWL Ontology Extraction from Relational Schema

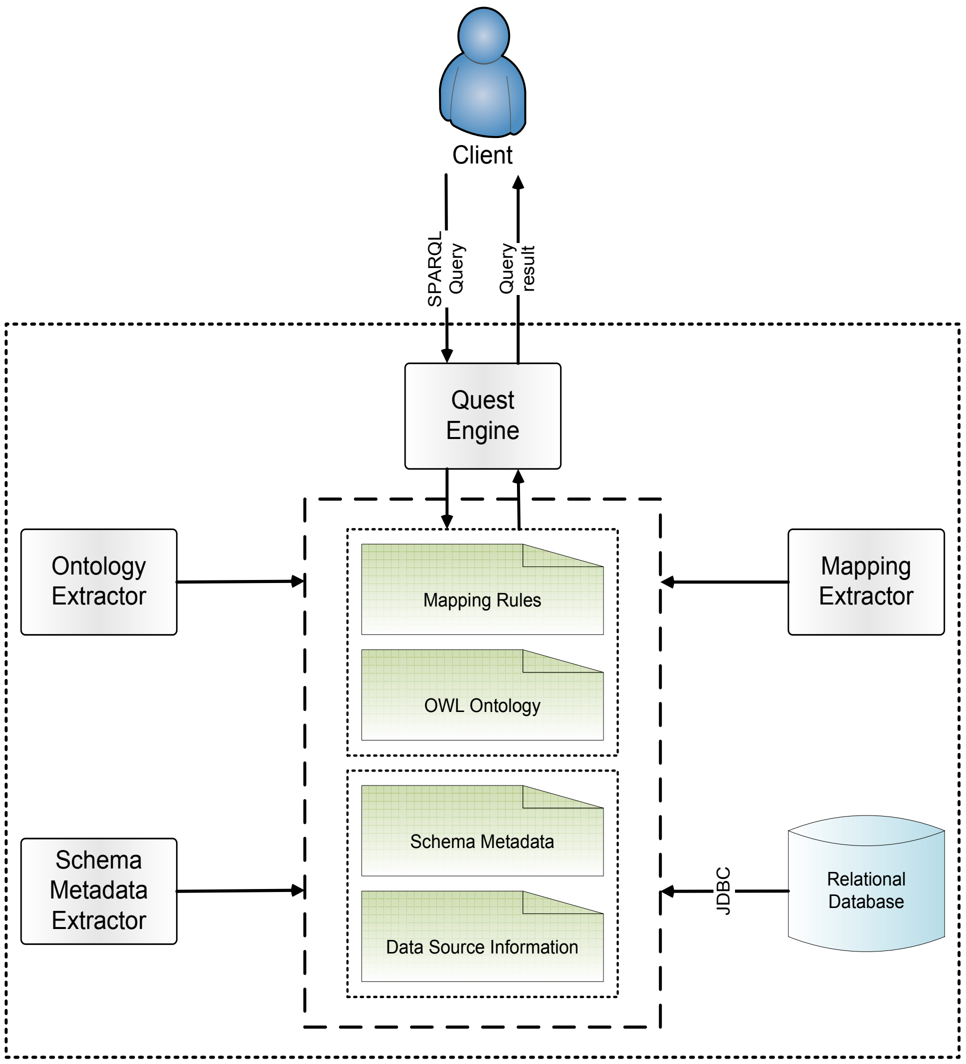


Figure 3.1: The architecture of the proposed approach.

In this section, we show our approach for extracting both the *Ontop* mapping rules, and the equivalent OWL ontology from the schema of a relational data source. The extraction process is composed of three modules: *Schema Metadata Extractor* module that uses a *Connection Wrapper* (implemented through JDBC API (Fisher et al., 2003)) to extract the definition of the relational schema (i.e., the SQL DDL details), *Ontop Mappings Extractor* (*OMsE*) module that uses the schema metadata to extract the required *Ontop* mapping rules to enable end users from accessing the given relational schema through *SPARQL* queries, and the *OWL Ontology Extractor* (*OOE*) module that depends on the metadata and *Ontop* mapping rules extracted from the second module to generate the equivalent OWL ontology for the given relational schema. *OMsE* module takes the description of the schema (as SQL DDL) and extracts *Ontop* mappings rules. The proposed approach is built on top of the *Quest* inference system as shown in Fig. 3.1.

Before we show our approach for extracting *Ontop* mappings and OWL ontology from a relational database schema, we define the terms used hereafter:

3.1 The Metadata of a Relational Schema

- Σ: A metadata of a relational schema that represents the entities and their relationships in a particular domain. It is defined as Σ: $\{\mathcal{T}_1, \mathcal{T}_2, ..., \mathcal{T}_t\}$, where $\mathcal{T}_i$ is a particular table/entity in the schema and t is the total number of tables in the schema.

- $\mathcal{T}_i$: $\{\mathcal{N}_i, \mathcal{A}_i, \mathcal{PK}_i, \mathcal{FK}_i, \mathcal{NK}_i\}$, where $\mathcal{N}_i$ is the name of the table, $\mathcal{A}_i$ is the set of all $\mathcal{T}_i$'s columns, $\mathcal{PK}_i$ is the set of $\mathcal{T}_i$'s primary key columns, $\mathcal{FK}_i$ is the set of $\mathcal{T}_i$'s foreign key columns, and $\mathcal{NK}_i$ is the set of T_i's non primary key columns. Therefore, $\mathcal{PK}_i \bigcap \mathcal{NK}_i = \phi$.

- $\mathcal{A}_i$: $\{a_1{:}d_1, a_2{:}d_2, ..., a_n{:}d_n\}$, where n is the number of columns in $\mathcal{T}_i$ and $a_j{:}d_j$ is the pair of column j's name (a_j) and the SQL data type (d_j).

- $\mathcal{PK}_i$ is either a single column key or a composite key, such that $\mathcal{PK}_i = \{\mathcal{K}_{P_i^1}, \mathcal{K}_{P_i^2}, \ldots, \mathcal{K}_{P_i^n}\}$, where $\mathcal{K}_{P_i^t}$ is the column t in $\mathcal{PK}_i$ and n is the total number of columns that are composing $\mathcal{PK}_i$.

- $\mathcal{FK}_i = \{\mathcal{K}_{F_i^1}, \mathcal{K}_{F_i^2}, \ldots, \mathcal{K}_{F_i^m}\}$, where $\mathcal{K}_{F_i^t}$ is the foreign key t in $\mathcal{FK}_i$ and m is the number of foreign keys in $\mathcal{T}_i$.

- $\mathcal{K}_{F_i^x} = \{\mathcal{K}_{f^1}, \mathcal{K}_{f^2}, \ldots, \mathcal{K}_{f^v}\}$, where $\mathcal{K}_{f^t}$ is the column t in $\mathcal{K}_{F_i^x}$ and v is the number of columns in foreign key $\mathcal{K}_{F_i^x}$.

- $\mathcal{PF}_i$ is the list of some foreign keys in $\mathcal{T}_i$, such that for each foreign key $\mathcal{K}_{F_i^t}$ in $\mathcal{PF}_i$, all columns in this foreign key are also in the primary key.

- $\mathcal{PFL}_i$ is the list of all columns in $\mathcal{T}_i$ that are also in the primary key list $\mathcal{PK}_i$.

3.2 Rules for extracting Ontop mapping rules from relational schema

In this section, we describe the *Ontop* Mappings Extractor (*OMsE*) mechanisms to extract the *Ontop* mappings ($\mathcal{M}$) from a relational schema.

Assuming we have the *SQL DDL* for a relational schema Σ as an input to the *OMsE*, Algorithm 3.2.1 illustrates the extraction process:

Definition 1. Independent Table. Let $\mathcal{T}_i$ be a table that has a primary key $\mathcal{PK}_i$. If the foreign keys set $\mathcal{FK}_i$ of the table $\mathcal{T}_i$ is *empty*, we say that $\mathcal{T}_i$ is an *independent table*

3.2.1 Independent Table rule

If a table has a primary key and no foreign keys ($\mathcal{PK} \neq \phi$ && $\mathcal{FK} = \phi$), we apply the *independent table* rule by calling the extraction of *Ontop* mapping rules algorithm in

ALGORITHM 3.2.1: Extracting Ontop Mapping rules from a relational schema.

```
input  : $\Sigma = \{T_1, T_2, ..., T_t\}$
output: $\mathcal{M}$
1  foreach $T_i \in \Sigma$ do
2      if $\mathcal{PK}_i \neq \phi$ then
3          if $\mathcal{FK}_i = \phi$ then
4              EOMRsForIndependentTable($T_i$, $\mathcal{M}$) ;
5          else if $size(\mathcal{PF}_i) \leq 1$ then
6              EOMRsForDependentTable($T_i$, $\mathcal{M}$) ;
7          else if $size(\mathcal{PF}_i) = 2$ then
8              if $\mathcal{NK}_i = \phi$ then
9                  EOMRsForBinJWithNoNKs($T_i$, $\mathcal{M}$) ;
10             else
11                 EOMRsForNaryJoinTable($T_i$, $\mathcal{M}$) ;
12         else if $size(\mathcal{PF}_i) > 2$ then
13             EOMRsForNaryJoinTable($T_i$, $\mathcal{M}$) ;
14         else if $size(\mathcal{FK}_i) \geq 1$ then
15             EOMRsForDependentTable($T_i$, $\mathcal{M}$) ;
16     else if $size(\mathcal{FK}_i) \geq 1$ then
17         EOMRsForDependentTable($T_i$, $\mathcal{M}$) ;
18     else
19         EOMRsForIsolatedTable($T_i$, $\mathcal{M}$) ;
20         Note: This rule is to represent an isolated table with neither a primary key nor a
           foreign key. We did not implement this rule in our approach because such a table
           is not an important.
```

Line 4 of Algorithm 3.2.1. The process of extracting an *Ontop* mapping rule from an independent table is shown in Algorithm 3.2.2. The algorithm takes the metadata of the independent table T_i and $\mathcal{M}$ as inputs. It then generates an *Ontop* mapping rule for the given table and adds it to $\mathcal{M}$.

ALGORITHM 3.2.2: Extracting Ontop Mapping Rules from an Independent Table.

```
Function EOMRsForIndependentTable($T_i$, $\mathcal{M}$)
    $OntopRule \leftarrow$ GenerateOntopMappingRule($T_i$) ;
    $sub \leftarrow$ GenerateSubject($T_i$, null, null, null);
    $triple \leftarrow$ GenerateClassTriple($T_i$, sub);
    $OntopRule.target \leftarrow$ Add(triple, "ClassTriple");
    foreach $a_j \in T_i$ do
        $obj \leftarrow$ GenerateDataPropertyObject($T_i$, $a_j$);
        $predicate \leftarrow$ GenerateDataPropertyPredicate($T_i$, $a_j$);
        $triple \leftarrow$ GenerateDataPropertyTriple($T_i$, sub, predicate, obj);
        if $a_j$ is not last attribute in $T_i$ then
            $triple \leftarrow triple +$ " ; ";
        else
            $triple \leftarrow triple +$ " . ";
        $OntopRule.target \leftarrow$ Add(triple, "DataPropertyTriple") ;
    $OntopRule.source \leftarrow$ GenerateSqlQuery($T_i$, null, null) ;
    $\mathcal{M} \leftarrow \mathcal{M} \cup OntopRule$ ;
    return $\mathcal{M}$;
```

We can represent the target part of the *independent table* rule by a class triple template and a data property template for each column that belongs to $\mathcal{T}_i$ as shown in (3.1).

$$\begin{aligned}\boldsymbol{TARGET:}\ \underline{:\mathcal{T}_i/\mathcal{K}_{P_i^1} = \{\mathcal{K}_{P_i^1}\}\ [\ ;\ \mathcal{K}_{P_i^t} = \{\mathcal{K}_{P_i^t}\}^{t=2..i}\ ]}\ \underline{a}\ \underline{:\mathcal{T}_i}\ ;\\ \underline{:\mathcal{T}_i\#a_1}\ \underline{\{\mathcal{T}_i.a_1\}}\ ;\\ \underline{:\mathcal{T}_i\#a_2}\ \underline{\{\mathcal{T}_i.a_2\}}\ ;\\ \dots\ \dots\ ;\\ \underline{:\mathcal{T}_i\#a_n}\ \underline{\{\mathcal{T}_i.a_n\}}\ .\end{aligned} \tag{3.1}$$

$$\boldsymbol{SOURCE:}\ SELECT\ *\ FROM\ \mathcal{T}_i \tag{3.2}$$

The class triple template has the form $subject(:\mathcal{T}_i \rightarrow \mathcal{T}_i)\ a : \mathcal{T}_i$ as shown in the first line of (3.1). $subject(:\mathcal{T}_i \rightarrow \mathcal{T}_i)$ maps class $:\mathcal{T}_i$'s data property attributes that represent $:\mathcal{T}_i$'s primary key with the corresponding columns of table $\mathcal{T}_i$. This term represents the subject of each triple of *Ontop* mapping rule's target query. The second term, a, is the predicate of the class triple template, and it is simply a syntactical shortcut for $rdf\!:\!type$. The last term is the object that represents the class type for each individual that has a subject that matches with the subject of the class triple template.

Each one of the remaining lines in (3.1) represents a data property triple template to map a data property attribute of class $:\mathcal{T}_i$ with its correspondent column in table $\mathcal{T}_i$. Here, we omit the subject of each data property triple because *Ontop* mapping rules are declared using the *turtle* format. *Turtle* is fast becoming the de facto standard for representing *RDF* files. In *turtle*, if the same subject is repeated many times, but with different predicates, we may separate each pair of predicate and object by semicolons. In the same way, if both subject and predicate are repeated, we may separate objects by commas. n in the last line of (3.1) represents the number of data attributes that class$:\mathcal{T}_i$ has or the number of column that are in table $\mathcal{T}_i$. The source query part of *Ontop* mapping rule for *Independent Table* is shown in (3.2). The target part of *independent table* rule are generated by Lines 2 to 14 of Algorithm 3.2.2, whereas the source part is generated by calling the function *GenerateSqlQuery* in Line 15.

Independent Table Example. We explain Algorithm 3.2.2 through this example. Assume we have a table, *dept*, as shown in Listing 3.1, that stores information about departments in an organization. The primary key of *dept* is the department number field (*deptno*). It also has the department name (*dname*) and department location (*loc*) fields. Since *dept* has only a primary key constraint and no foreign key constraints, the *independent table* rule applies to *dept*. Thus, Algorithm 3.2.2 that extracts an *Ontop* mappings from an independent table is executed in Line 4 of Algorithm 3.2.1. Listing 3.2 shows the extracted *Ontop* mapping rule for *dept*. The resulting target part of the rule is according to the target triple template in (3.1) and the resulting source part of the rule is according to the source template in (3.2).

```
CREATE TABLE dept
(
    deptno int(2),
    dname varchar(14) NOT NULL,
    loc varchar(13) NOT NULL,
    CONSTRAINT pk_dept PRIMARY KEY (deptno)
);
```

Listing 3.1: Independent Table sample schema.

```
target      :Dept/deptno={deptno} a :Dept ;
            :Dept#deptno {deptno}^^xsd:integer ;
            :Dept#dname {dname} ; :Dept#loc {loc} .
source      SELECT * FROM dept
```

Listing 3.2: Ontop Mapping rule for the Independent Table in Listing 3.1.

Definition 2. Dependent Table. Let $\mathcal{T}_i$ be a table in Σ. If (1) $\mathcal{PK}_i \neq \phi$ and $|\mathcal{PF}_i| \leq 1$, or (2) $\mathcal{PK}_i \neq \phi$ and $\mathcal{PF}_i = \phi$ and $|\mathcal{FK}_i| \geq 1$, or (3) $\mathcal{PK}_i = \phi$ and $|\mathcal{FK}_i| \geq 1$, we say that $\mathcal{T}_i$ is a *dependent table*.

3.2.2 Dependent Table rule

If a table has one or more foreign keys that reference other tables and one of the three cases in the definition above applies, then there is a one-to-many relationship between each referenced table and this table. In this case, we apply the *dependent table* rule by calling the extracting of *Ontop* mapping rules algorithm in Lines 6, 15, and 17 of Algorithm 3.2.1.

The process of extracting *Ontop* mapping rules from a dependent table is shown in Algorithm 3.2.3. The algorithm takes the metadata of the dependent table T_i and $\mathcal{M}$ as inputs. It then generates the *Ontop* mapping rules for the given table and adds it to $\mathcal{M}$. Algorithm 3.2.3 generates an *Ontop* mapping rule for the dependent table the same way as in the independent table. It also generates an *Ontop* mapping rule for each foreign key in the dependent table. However, it applies the *recursive table* rule if a particular foreign key is a self reference to the same table. The *recursive table* rule is applied by calling the extracting of *Ontop* mapping rules algorithm in Line 8 of Algorithm 3.2.3.

ALGORITHM 3.2.3: Extracting Ontop Mapping Rules from a Dependent Table.

```
1  Function EOMRsForDependentTable($T_i$, $\mathcal{M}$)
2      $\mathcal{M} \leftarrow$ EOMRsForIndependentTable($T_i$, $\mathcal{M}$) ;
3      $sub_i \leftarrow$ GenerateSubject($T_i$, $null$, $null$, $null$);
4      foreach $\mathcal{K}_F \in \mathcal{FK}_i$ do
5          if ($T_i.nAryJnTable = false$) or ($T_i.nAryJnTable = true$ and $\mathcal{K}_F \notin \mathcal{PF}_i$) then
6              $T_j \leftarrow$ GetReferencedTable($\mathcal{K}_F$);
7              if $T_j = T_i$ then
8                  EOMRsForRecursiveTable($T_i$, $\mathcal{M}$, $\mathcal{K}_F$) ;
9              else
10                 $OP_j \leftarrow$ CreateObjectPropertyName($T_j.name$, "$has$", $T_i.name$);
11                 $OP_i \leftarrow$ CreateObjectPropertyName($T_i.name$, "$has$", $T_j.name$);
12                 $OntopRule \leftarrow$ GenerateOntopMappingRule($T_i$, $\mathcal{K}_F$) ;
13                 $sub_j \leftarrow$ GenerateSubject($T_j$, $null$, "$DEPENDENT$", $null$);
14                 $predicate_i \leftarrow$ GenerateObjectPropertyPredicate($T_i$, $OP_i$);
15                 $predicate_j \leftarrow$ GenerateObjectPropertyPredicate($T_j$, $OP_j$);
16                 $triple \leftarrow$ GenerateObjectPropertyTriple($T_i$, $sub_i$, $predicate_i$, $sub_j$);
17                 $OntopRule.target \leftarrow$ Add($triple$, "$ObjectPropertyTriple$");
18                 $triple \leftarrow$ GenerateObjectPropertyTriple($T_j$, $sub_j$, $predicate_j$, $sub_i$);
19                 $OntopRule.target \leftarrow$ Add($triple$, "$ObjectPropertyTriple$") ;
20                 $OntopRule.source \leftarrow$ GenerateSqlQuery($T_i$, "$DEPENDENT$", $\mathcal{K}_F$) ;
21                 $\mathcal{M} \leftarrow \mathcal{M} \cup OntopRule$ ;
22     return $\mathcal{M}$;
```

There are two steps for extracting *Ontop* mapping rules from $\mathcal{T}_i$ that satisfies the *dependent table* rule. The first step is to extract an *Ontop* mapping rule as we do in *independet table* rule. The second step is to extract an *Ontop* mapping rule for each foreign key in $\mathcal{T}_i$. We skip the first step because it is the same as in the previous rule. The details of the second step follow.

TARGET **:**

$$\underline{:\mathcal{T}_i/\mathcal{K}_{P_i^1} = \{\mathcal{T}_i\_\mathcal{K}_{P_i^1}\}\ [\ ;\ \mathcal{K}_{P_i^t} = \{\mathcal{T}_i\_\mathcal{K}_{P_i^t}\}\ ]^{t=2..i}}$$
$$\underline{:\mathcal{T}_i\#has\mathcal{T}_j}$$
$$\underline{:\mathcal{T}_j/\mathcal{K}_{P_j^1} = \{\mathcal{T}_j\_\mathcal{K}_{P_j^1}\}\ [\ ;\ \mathcal{K}_{P_j^t} = \{\mathcal{T}_j\_\mathcal{K}_{P_j^t}\}\ ]^{t=2..j}}\ . \tag{3.3}$$

$$\underline{:\mathcal{T}_j/\mathcal{K}_{P_j^1} = \{\mathcal{T}_j\_\mathcal{K}_{P_j^1}\}\ [\ ;\ \mathcal{K}_{P_j^t} = \{\mathcal{T}_j\_\mathcal{K}_{P_j^t}\}\ ]^{t=2..j}}$$
$$\underline{:\mathcal{T}_j\#has\mathcal{T}_i}$$
$$\underline{:\mathcal{T}_i/\mathcal{K}_{P_i^1} = \{\mathcal{T}_i\_\mathcal{K}_{P_i^1}\}\ [\ ;\ \mathcal{K}_{P_i^t} = \{\mathcal{T}_i\_\mathcal{K}_{P_i^t}\}\ ]^{t=2..i}}\ . \tag{3.4}$$

SOURCE **:**

$$\begin{array}{ll} \textit{SELECT} & \mathcal{T}_i.\mathcal{K}_{P_i^1}\ AS\ \mathcal{T}_i\_\mathcal{K}_{P_i^1}\ [\ ,\ \mathcal{T}_i.\mathcal{K}_{P_i^t}\ AS\ \mathcal{T}_i\_\mathcal{K}_{P_i^t}\ ]^{t=2..i}\ , \\ & \mathcal{T}_j.\mathcal{K}_{P_j^1}\ AS\ \mathcal{T}_j\_\mathcal{K}_{P_j^1}\ [\ ,\ \mathcal{T}_j.\mathcal{K}_{P_j^t}\ AS\ \mathcal{T}_j\_\mathcal{K}_{P_j^t}\ ]^{t=2..j} \\ \textit{FROM} & \mathcal{T}_i\ ,\ \mathcal{T}_j\ WHERE\ \mathcal{T}_i.\mathcal{K}_{f_i^1}\ =\ \mathcal{T}_j.\mathcal{K}_{P_j^1}\ [\ AND\ \mathcal{T}_i.\mathcal{K}_{f_i^t}\ =\ \mathcal{T}_j.\mathcal{K}_{P_j^t}\ ]^{t=2..j} \end{array} \tag{3.5}$$

For each foreign key $\mathcal{K}_{F_i^t}$ in $\mathcal{T}_i$ that references a table $\mathcal{T}_j$, we extract an *Ontop* mapping rule with *target* and *source* parts. The *target* part is shown in (3.3) and (3.4). It has two object property triple templates. The first triple represents the many-to-one relation from $\mathcal{T}_i$ to $\mathcal{T}_j$. The other triple is simply the inverse of first one. In other words, it represents the one-to-many relation from $\mathcal{T}_j$ to $\mathcal{T}_i$. Thus, the predicate of the second triple is the inverse of the predicate of the first one.

The format of the triple template (3.3) is $subject(:\mathcal{T}_i \rightarrow \mathcal{T}_i) : \mathcal{T}_i\#has\mathcal{T}_j\ subject(:\mathcal{T}_j \rightarrow \mathcal{T}_j)$. $subject(:\mathcal{T}_i \rightarrow \mathcal{T}_i)$ maps class $:\mathcal{T}_i$'s individuals with their correspondent rows resulted from the join query of the source that come from table $\mathcal{T}_i$, and $subject(:\mathcal{T}_j \rightarrow \mathcal{T}_j)$ maps class $:\mathcal{T}_j$'s individuals with their correspondent rows resulted from the join query of the source that come from table $\mathcal{T}_j$. The format of the triple template (3.4) is the same as (3.3), except it represents the mapping from the opposite direction.

The target part of an *Ontop* mapping rule for each foreign key of *dependent table* that references a table $\mathcal{T}_j$ is generated by Lines 12 to 19 of Algorithm 3.2.3. The source query part of an *Ontop* mapping rule for each foreign key of *dependent table* that references a table $\mathcal{T}_j$ is shown in (3.5). It is generated by calling the function *GenerateSqlQuery* in Line 20 of Algorithm 3.2.3. *GenerateSqlQuery* is shown in Algorithm E.0.1. Based on the value of the second parameter that is passed to the function by Algorithm 3.2.3 (i.e., “DEPENDENT”), the function *GenerateSqlQuery-ForDependentRule* is called in Line 7 to generate a source query that is compatible with the *Ontop* mapping rule for a foreign key of *dependent table*.

Dependent Table Example. We explain Algorithm 3.2.3 through the following example. Assume we have two tables: *dept* and *emp*. The first one is the same table that is used in the previous example. The second table, *emp*, as shown in Listing 3.3, stores information about employees in an organization. The primary key of *emp* is the employee number field (*empno*), with the employee name (*empname*), job (*job*), hire date (*hiredate*), and salary (*sal*) fields. It also has the department number field (*deptno*) that represents a foreign key that references *deptno* (the primary key of table *dept*). In addition, it has the manager field (*mgr*) that represents a self foreign key reference to the same table.

```sql
CREATE TABLE emp
(
    empno int(4),
    empname varchar (10) NOT NULL,
    job varchar(9) NOT NULL,
    mgr int(4),    -- can be null for the president.
    hiredate date NOT NULL,
    sal double(7, 2) NOT NULL,
    deptno int(2) NOT NULL,
    CONSTRAINT pk_emp PRIMARY KEY (empno),
    CONSTRAINT fk_deptno FOREIGN KEY (deptno) REFERENCES dept(deptno),
    CONSTRAINT fk_mgr  FOREIGN KEY (mgr) REFERENCES emp(empno)
);
```

Listing 3.3: Dependent and Recursive tables sample schema.

Since *emp* has a primary key constraint and some foreign key constraints, the *dependent table* rule applies on *emp*. Thus, Algorithm 3.2.3 that extracts an *Ontop* mappings from a *dependent table* is executed in Line 6 of Algorithm 3.2.1. The first step in Algorithm 3.2.3 is to extract an *Ontop* mapping rule for table *emp* the same way as in the *independent table* by calling Algorithm 3.2.2 in Line 2. The outcome of this step is the *Ontop* mapping rule shown in Listing 3.4. Next, we extract an *Ontop* mapping rule for each foreign key in *emp*. The table *emp* has two foreign keys fields: *mgr* and *deptno*. The first one is a recursive reference to *emp*. Thus, we apply here the *recursive* rule to extract the *Ontop* mapping rule that represents the self-reference by calling Algorithm 3.2.4 in Line 8. We leave the details of extracting the *recursive* rule to the next example. The *Ontop* mapping rule for the foreign key *deptno* is then extracted. The resulted *Ontop* mapping rule is shown in Listing 3.5. The target part of the rule is according Templates (3.3) and (3.4). The source part is according to Template 3.5.

```
target       :Emp/empno={empno} a :Emp ;
             :Emp#empno {empno}^^xsd:integer ;
             :Emp#empname {empname} ;
             :Emp#job {job} ;
             :Emp#mgr {mgr}^^xsd:integer ;
             :Emp#hiredate {hiredate} ;
             :Emp#sal {sal}^^xsd:double ;
             :Emp#deptno {deptno}^^xsd:integer .
source       SELECT * FROM emp
```

Listing 3.4: First Ontop Mapping rule for Dependent table in Listing 3.3.

```
target       :Emp/empno={emp_empno} :Emp#hasDept :Dept/deptno={dept_deptno} .
             :Dept/deptno={dept_deptno} :Dept#hasEmp empno={emp_empno} .
source       SELECT emp.empno AS emp_empno, dept.deptno AS dept_deptno
             FROM emp, dept WHERE emp.deptno = dept.deptno
```

Listing 3.5: Second Ontop Mapping rule for Dependent table in Listing 3.3.

Definition 3. Recursive Table. Let $\mathcal{T}_i$ be a table that has a primary key $\mathcal{PK}_i$. Let also $\mathcal{K}_F$ to be a foreign key in table $\mathcal{T}_i$, such that it references the same table (i.e., recursive). If there is at least one $\mathcal{K}_F$ that is a self reference on the table $\mathcal{T}_i$, we say that this table is a *recursive table*.

3.2.3 Recursive Table rule

If a table has a foreign key that references itself, then there is a recursive relationship between this foreign key and its own table. As discussed above, the *recursive table* rule is applied as a sub-case of the *dependent table* rule. Extracting the *Ontop* mapping rules from a recursive reference is shown in Algorithm 3.2.4. The algorithm takes the metadata of the recursive table T_i, $\mathcal{M}$, and the recursive foreign key reference $\mathcal{K}_F$ as inputs. It then generates the recursive *Ontop* mapping rule for the given table and adds it to $\mathcal{M}$.

ALGORITHM 3.2.4: Extracting Ontop Mapping Rule from a Recursive Table.

```
Function EOMRsForRecursiveTable($T_i$, $\mathcal{M}$, $\mathcal{K}_F$)
    $OP_i \leftarrow$ CreateObjectPropertyName($T_i.name$, “has”, $T_i.name$);
    $OntopRule \leftarrow$ GenerateOntopMappingRule($T_i$, $\mathcal{K}_F$) ;
    $sub_i \leftarrow$ GenerateSubject($T_i$, null, “RECURSIVE”, “DOMAIN”);
    $sub_j \leftarrow$ GenerateSubject($T_i$, null, “RECURSIVE”, “RANGE”);
    $predicate_i \leftarrow$ GenerateObjectPropertyPredicate($T_i$, $OP_i$);
    $triple \leftarrow$ GenerateObjectPropertyTriple($T_i$, $sub_i$, $predicate_i$, $sub_j$);
    $OntopRule.target \leftarrow$ Add(triple, “ObjectPropertyTriple”) ;
    $OntopRule.source \leftarrow$ GenerateSqlQuery($T_i$, “RECURSIVE”, $\mathcal{K}_F$) ;
    $\mathcal{M} \leftarrow \mathcal{M} \cup OntopRule$ ;
    return $\mathcal{M}$;
```

For each recursive foreign key $\mathcal{K}_F$ in $\mathcal{T}_i$, we extract an *Ontop* mapping rule with *target* and *source* parts. The *target* part is shown in (3.6). It has one object property triple template. This triple represents the recursive relation on $\mathcal{T}_i$ that comes from the recursive reference $\mathcal{K}_F$. The format of the triple template (3.6) is $subject(:\mathcal{T}_i \rightarrow \mathcal{T}_i) : \mathcal{T}_i\#has\mathcal{T}_i\ object(:\mathcal{T}_i \rightarrow \mathcal{T}_i)$. The target part of an *Ontop* mapping rule for a recursive foreign key in $\mathcal{T}_i$ is generated by Lines 3 to 8 of Algorithm 3.2.4. The source query part of an *Ontop* mapping rule for a recursive foreign key in $\mathcal{T}_i$ is shown in (3.7). It is generated by calling the function *GenerateSqlQuery* in Line 9 of Algorithm 3.2.4. *GenerateSqlQuery* is shown in Algorithm E.0.1. Based on the value of the second parameter that is passed to the function by Algorithm 3.2.3 (i.e., “RECURSIVE”), the function *GenerateSqlQueryForRecursiveRule* is called in Line 9 to generate a source query that is compatible with the *Ontop* mapping rule for a self reference.

***TARGET* :**

$$\underline{:\mathcal{T}_i/\mathcal{K}_{P_i^1} = \{\mathcal{T}_i\_child\_\mathcal{K}_{P_i^1}\}\ [\ ;\ \mathcal{K}_{P_i^t} = \{\mathcal{T}_i\_child\_\mathcal{K}_{P_i^t}\}\ ]^{t=2..i}}$$
$$\underline{:\mathcal{T}_i\#has\mathcal{T}_i}$$
$$\underline{:\mathcal{T}_i/\mathcal{K}_{P_i^1} = \{\mathcal{T}_i\_parent\_\mathcal{K}_{P_i^1}\}\ [\ ;\ \mathcal{K}_{P_i^t} = \{\mathcal{T}_i\_parent\_\mathcal{K}_{P_i^t}\}\ ]^{t=2..i}}\ . \qquad (3.6)$$

SOURCE :

$$\begin{aligned}
&SELECT\ \mathcal{T}_i\_child.\mathcal{K}_{P_i^1}\ AS\ \mathcal{T}_i\_child\_\mathcal{K}_{P_i^1}\ [\ ,\ \mathcal{T}_i\_child.\mathcal{K}_{P_i^t}\ AS\ \mathcal{T}_i\_child\_\mathcal{K}_{P_i^t}\ ]^{t=2..i}\ ,\\
&\qquad \mathcal{T}_i\_parent.\mathcal{K}_{P_i^1}\ AS\ \mathcal{T}_i\_parent\_\mathcal{K}_{P_i^1}\ [\ ,\ \mathcal{T}_i\_parent.\mathcal{K}_{P_i^t}\ AS\ \mathcal{T}_i\_parent\_\mathcal{K}_{P_i^t}\ ]^{t=2..i}\\
&FROM\ \mathcal{T}_i\ \mathcal{T}_i\_child\ ,\ \mathcal{T}_i\ \mathcal{T}_i\_parent\\
&WHERE\ \mathcal{T}_i\_child.\mathcal{K}_{f^1}\ =\ \mathcal{T}_i\_parent.\mathcal{K}_{P_i^1}\ [\ AND\ \mathcal{T}_i\_child.\mathcal{K}_{f^t}\ =\ \mathcal{T}_i\_parent.\mathcal{K}_{P_i^t}\ ]^{t=2..i}
\end{aligned} \tag{3.7}$$

<u>Recursive Table Example</u>. We explain Algorithm 3.2.4 through this example. We go back to the previous example and consider table *emp* shown in Listing 3.3. The field *mgr* in *emp* is a foreign key that references *emp* itself. Thus, we apply the *recursive table* rule by calling Algorithm 3.2.4. The outcome is the *Ontop* mapping rule shown in Listing 3.6. The target part of the rule is resulted by applying Template (3.6) and the source part is resulted by applying the query Template (3.7).

```
target       :Emp/empno={emp_child_empno} :Emp#hasEmp :Emp/empno={emp_parent_empno} .
source       SELECT emp_child.empno AS emp_child_empno,
                    emp_parent.empno AS emp_parent_empno
             FROM emp emp_child, emp emp_parent
             WHERE emp_child.mgr = emp_parent.empno
```

Listing 3.6: Ontop Mappings for Recursive table in Listing 3.3.

Definition 4. <u>Binary Join Table</u>. Let $\mathcal{T}_i$, $\mathcal{T}_j$, and $\mathcal{T}_k$ be three tables with primary keys $\mathcal{PK}_i$, $\mathcal{PK}_j$, and $\mathcal{PK}_k$, respectively. If (1) the primary key of $\mathcal{T}_i$ is composed of two parts ($\mathcal{PF}_{ij}$ and $\mathcal{PF}_{ik}$), where the former is both the first part of $\mathcal{PK}_i$ and the foreign key that references the primary key of $\mathcal{T}_j$, and the latter is both the second part of $\mathcal{PK}_i$ and the foreign key that references the primary key of $\mathcal{T}_k$, and (2) all $\mathcal{T}_i$'s columns are in the primary key, we say that $\mathcal{T}_i$ is a $binary-join\ table$.

3.2.4 Binary Join Table rule

If a table $\mathcal{T}_i$ has a composite primary key of two foreign keys (that reference tables $\mathcal{T}_j$ and $\mathcal{T}_k$) and all $\mathcal{T}_i$'s columns are in the primary key and are also in one of the two foreign keys ($\mathcal{PK}_i = \mathcal{PFL}_i$, $|\mathcal{FK}_i| = 2$, and $\mathcal{NK}_i = \phi$), then this table represents a binary relation with non primary key columns that connects $\mathcal{T}_j$ and $\mathcal{T}_k$ together in a many-to-many relationship. This binary relation can be divided into two sub relations; one one-to-many sub-relation from $\mathcal{T}_j$ to $\mathcal{T}_k$, and another one-to-many sub-relation from $\mathcal{T}_k$ to $\mathcal{T}_j$.

ALGORITHM 3.2.5: Extract Ontop Mapping Rules for Binary-Join Table with no non-key attributes.

```
Function EOMRsForBinJWithNoNKs($T_i$, $\mathcal{M}$)
    $OntopRule \leftarrow$ GenerateOntopMappingRule() ;
    foreach $\mathcal{K}_F \in \mathcal{FK}_i$ do
        $\mathcal{K}_{F_{other}} \leftarrow \mathcal{FK}_i$ - $\mathcal{K}_F$ ;
        $T_j \leftarrow$ GetReferencedTable($\mathcal{K}_F$);
        $T_k \leftarrow$ GetReferencedTable($\mathcal{K}_{F_{other}}$);
        $sub_j \leftarrow$ GenerateSubject($T_j$, $T_i$, "BINARYJOIN", null);
        $sub_k \leftarrow$ GenerateSubject($T_k$, $T_i$, "BINARYJOIN", null);
        $OP_j \leftarrow$ CreateObjectPropertyName($T_j.name$, "has_", $T_k.name$);
        $predicate_j \leftarrow$ GenerateObjectPropertyPredicate($T_j$, $OP_j$);
        $triple \leftarrow$ GenerateObjectPropertyTriple($T_j$, $sub_j$, $predicate_j$, $sub_k$);
        $OntopRule.target \leftarrow$ Add(triple, "ObjectPropertyTriple") ;
    $OntopRule.source \leftarrow$ GenerateSqlQuery($T_i$, null) ;
    $\mathcal{M} \leftarrow \mathcal{M} \cup OntopRule$ ;
    return $\mathcal{M}$;
```

Thus, $\mathcal{T}_i$ represents a binary relationship table between the two tables $\mathcal{T}_j$ and $\mathcal{T}_k$. We can represent this kind of binary relation in ontology without adding an equivalent class entity for table $\mathcal{T}_i$. Instead, we add one object property for each one-to-many relationship. The first object property has the extracted concept of $\mathcal{T}_j$ as its domain and the extracted concept of $\mathcal{T}_k$ as its range. The second object property has the extracted concept of $\mathcal{T}_k$ as its domain and the extracted concept of $\mathcal{T}_j$ as its range. In other words, each object property is simply the inverse of the other.

In this case, we apply the $binary-join\ table$ rule by calling the extraction of *Ontop* mapping rules algorithm in Line 9 of Algorithm 3.2.1. The process of extracting an *Ontop* mapping rule from a binary join table is shown in Algorithm 3.2.5. The algorithm takes the metadata of the independent table T_i and $\mathcal{M}$ as inputs. It then generates an *Ontop* mapping rule for the given table and adds it to $\mathcal{M}$.

$$\boldsymbol{TARGET:}$$

$$\begin{array}{l} \underline{:\mathcal{T}_j/\mathcal{K}_{P^1_j} = \{\mathcal{K}_{PF^1_j}\}\ [\ ;\ \mathcal{K}_{P^t_j} = \{\mathcal{K}_{PF^t_j}\}\ ]^{t=2..j}} \\ \qquad\qquad \underline{:\mathcal{T}_j\#has\mathcal{T}_k} \\ \qquad\qquad\qquad\qquad \underline{:\mathcal{T}_k/\mathcal{K}_{P^1_k} = \{\mathcal{K}_{PF^1_k}\}\ [\ ;\ \mathcal{K}_{P^t_k} = \{\mathcal{K}_{PF^t_k}\}\ ]^{t=2..k}}\ . \end{array} \tag{3.8}$$

$$\begin{array}{l} \underline{:\mathcal{T}_k/\mathcal{K}_{P^1_k} = \{\mathcal{K}_{PF^1_k}\}\ [\ ;\ \mathcal{K}_{P^t_k} = \{\mathcal{K}_{PF^t_k}\}\ ]^{t=2..k}} \\ \qquad\qquad \underline{:\mathcal{T}_k\#has\mathcal{T}_j} \\ \qquad\qquad\qquad\qquad \underline{:\mathcal{T}_j/\mathcal{K}_{P^1_j} = \{\mathcal{K}_{PF^1_j}\}\ [\ ;\ \mathcal{K}_{P^t_j} = \{\mathcal{K}_{PF^t_j}\}\ ]^{t=2..j}}\ . \end{array} \tag{3.9}$$

$$\boldsymbol{SOURCE:}\ SELECT\ *\ FROM\ \mathcal{T}_i \tag{3.10}$$

We can represent the target part of $binary-join$ rule by using two triple templates. The first template has the form $subject(:\mathcal{T}_j \to \mathcal{T}_i)\ :\mathcal{T}_j\#has\mathcal{T}_k\ object(:\mathcal{T}_k \to \mathcal{T}_i)$ as shown in (3.8). $subject(:\mathcal{T}_j \to \mathcal{T}_i)$ maps class $:\mathcal{T}_j$'s data property attributes that represent $:\mathcal{T}_j$'s primary key with their correspondent columns of table $\mathcal{T}_i$. The same applies for $object(:\mathcal{T}_k \to \mathcal{T}_i)$. $:\mathcal{T}_j\#has\mathcal{T}_k$ represents the object property that maps an individual of class $:\mathcal{T}_j$'s with class $:\mathcal{T}_k$'s individuals. In other words, it performs the task as table $\mathcal{T}_i$ that joins both $\mathcal{T}_j$ and $\mathcal{T}_k$ in a binary relation. The second template has the form $subject(:\mathcal{T}_k \rightarrow \mathcal{T}_i) : \mathcal{T}_k\#has\mathcal{T}_j\ object(:\mathcal{T}_j \rightarrow \mathcal{T}_i)$ as shown in (3.9). It is simply the reverse of the first template. However, template triple (3.8) represents the one-to-many sub-relation from $\mathcal{T}_j$ to $\mathcal{T}_k$, and template triple (3.9) represents the one-to-many sub-relation from $\mathcal{T}_k$ to $\mathcal{T}_j$, as we mentioned above. The source query part of *Ontop* mapping rule for $binary - join$ is shown in (3.10), which is simply taken the same form as that of *independent table* rule. The target part of $binary - join$ rule are generated by Lines 2 to 12 of Algorithm 3.2.5, whereas the source part is generated by calling the function *GenerateSqlQuery* in Line 13.

Binary Join Table Example. We explain Algorithm 3.2.5 through this example. Assume we have three tables *employee*, *project* and *employee_project*, as shown in Listing 3.7. *employee_project* represents a binary join table that connects both *employee* and *project*. Both *employee* and *project* are independent tables. Thus, their *Ontop* mapping rules are extracted according to the *independent table* rules as shown in Listing 3.8. Table *employee_project* has a composed primary key of *emp_id* and *proj_id*, such that the former foreign key references table *employee* and the latter foreign key references table *project*. In other words, the primary key is only composed of these two foreign keys, such that each foreign key references another table. In addition, table *employee_project* does not have any fields other than the ones that are composing its primary key and both foreign keys that are representing the binary join. Thus, the $binary - join$ *table* rule applies here on *employee_project*. Hence, Algorithm 3.2.5 that extracts an *Ontop* mappings from a binary join table is called in Line 9 of Algorithm 3.2.1. The resulted *Ontop* mapping rule is shown in Listing 3.9. The target part of the rule is according Template (3.8) and (3.9). The source part is according to Template (3.10).

```sql
CREATE TABLE employee
(
     employee_id int NOT NULL,
     fname varchar(15) NOT NULL,
     lname varchar(15) NOT NULL,
    PRIMARY KEY (employee_id)
);

CREATE TABLE project
(
     project_id int NOT NULL,
     project_name varchar(15) NOT NULL,
    PRIMARY KEY (proj_id)
);

CREATE TABLE employee_project
(
     emp_id int NOT NULL,
     proj_id int NOT NULL,
    PRIMARY KEY (emp_id, proj_id),
    CONSTRAINT empid_fk FOREIGN KEY (emp_id) REFERENCES employee (employee_id),
    CONSTRAINT projid_fk FOREIGN KEY (proj_id) REFERENCES project (project_id)
);
```

Listing 3.7: An example on a relational schema for Binary relationship table without non-key columns rule.

```
target      :Employee/{employee_id} a :Employee ;
            :Employee#employee_id {employee_id}^^xsd:integer ;
            :Employee#fname {fname} ; :Employee#lname {lname} .
source      SELECT * FROM employee

target      :Project/{project_id} a :Project ;
            :Project#project_id {project_id}^^xsd:integer ;
            :Project#project_name {project_name} .
source      SELECT * FROM project
```

Listing 3.8: Ontop Mapping rules for the two tables (employee and project) that are part of the Binary Join Table employee-project.

```
target        :Project/{proj_id} :Project#hasEmployee  :Employee/{emp_id} .
              :Employee/{emp_id} :Employee#hasProject  :Project/{proj_id} .
source        SELECT * FROM employee_project
```

Listing 3.9: Ontop Mapping rule for Binary Join Table on Listing 3.7.

Definition 5. n-ary Join Table. Let $\mathcal{T}_i$ be a table that has a primary key $\mathcal{PK}_i$. Let $\mathcal{T}_1$, $\mathcal{T}_2$, ... , $\mathcal{T}_n$ be n tables with primary keys $\mathcal{PK}_1$, $\mathcal{PK}_2$, ..., $\mathcal{PK}_n$, respectively. If (1) the primary key of $\mathcal{T}_i$ is composed of n parts ($\mathcal{PF}_{i1}$, $\mathcal{PF}_{i2}$, ... , $\mathcal{PF}_{in}$), where the $1st$ part is the foreign key that references the primary key of $\mathcal{T}_1$, the $2nd$ part is the foreign key that references the primary key of $\mathcal{T}_2$, ... , and the nth part is the foreign key that references the primary key of $\mathcal{T}_n$, (2) either $n = 2$ and $\mathcal{NK}_i \neq \phi$ or $n > 2$, we say that $\mathcal{T}_i$ is an $n - ary\ join\ table$.

3.2.5 n-ary Join Table rule

This rule applies for an n-ary join table or (a binary-join table with non primary key columns). The first case is when a table $\mathcal{T}_i$ has a composite primary key of three or more foreign keys, such that each foreign key is referencing another table in Σ, we say that $\mathcal{T}_i$ connects three or more tables together in a many-to-many relationship. The second case is when a table $\mathcal{T}_i$ has a composite primary key of two foreign keys and it has some non primary key columns. In these two cases, we apply the $n-ary\ join\ table$ rule in Lines 11 and 13 of Algorithm 3.2.1. The difference between the binary join rule and the n-ary join rule is that in the former, only ontological object properties are extracted for representing the binary join table, while in the latter, an ontological class is extracted for representing the n-ary join table. In addition, we also apply the n-ary join rule on a binary join table with some non primary key columns as stated above.

ALGORITHM 3.2.6: Extracting Ontop Mapping Rules for Nary-Join Table or (Binary-Join Table with non primary key columns.)

```
1  Function EOMRsForNaryJoinTable($T_i$, $\mathcal{M}$)
2      $OntopRule \leftarrow$ GenerateOntopMappingRule($T_i$, "$NaryJoin$") ;
3      $sub_i \leftarrow$ GenerateSubject($T_i$, $null$, $null$, $null$);
4      $OP_i \leftarrow$ CreateObjectPropertyName($T_i.name$, "$has$", "$NaryJoin$");
5      $predicate_i \leftarrow$ GenerateObjectPropertyPredicate($T_i$, $OP_i$);
6      $triple \leftarrow null$ ;
7      foreach $\mathcal{K}_F \in \mathcal{PF}_i$ do
8          $triple \leftarrow triple \cup$ GenerateObjectPropertyTripleForNaryJoin($\mathcal{K}_F$, $sub_i$,
           $predicate_i$, $triple$);
9      $triple \leftarrow triple \cup$ " . " ;
10     $OntopRule.target \leftarrow$ Add($triple$, "$ObjectPropertyTriple$") ;
11     $OntopRule.source \leftarrow$ GenerateSqlQuery($T_i$, "$NARYJOIN$", $null$) ;
12     $\mathcal{M} \leftarrow \mathcal{M} \cup OntopRule$ ;
13
14     $\mathcal{T}_i.nAryJnTable = true$;
15     $\mathcal{M} \leftarrow$ EOMRsForDependentTable($T_i$, $\mathcal{M}$);
16     return $\mathcal{M}$;
```

The process of extracting *Ontop* mapping rules from an n-ary join table is shown in Algorithm 3.2.6. The algorithm takes the metadata of the n-ary join table T_i and $\mathcal{M}$ as inputs. It then generates the *Ontop* mapping rules for the given table and adds it to $\mathcal{M}$. Algorithm 3.2.6 generates an *Ontop* mapping rule for the n-ary join table the same way as in the dependent table rule. However, no *Ontop* mapping rule is extracted for any foreign key that belongs to the n-ary join relation. After that, it generates an *Ontop* mapping rule to represent the n-ary join relation that is composed of the foreign keys that are skipped in the previous step.

There are three steps for extracting *Ontop* mapping rules from $\mathcal{T}_i$ that satisfies the $n - ary\ join\ table$ rule. The first step is to extract an *Ontop* mapping rule as we do in *independet table* rule. The second step is to extract an *Ontop* mapping rule for each foreign key in $\mathcal{T}_i$ that does not belong to $\mathcal{PF}_i$. The last step is to extract an *Ontop* mapping rule from the foreign keys in $\mathcal{PF}_i$ which are representing the n-ary join relation in $\mathcal{T}_i$. We skip the first two steps because they are the same as in independent and dependent table rules. The details of the last step follow.

We extract an *Ontop* mapping rule with *target* and *source* parts to represent the n-ary join relation in $\mathcal{T}_i$. The *target* part is shown in (3.11). It has a number of object property triple templates that is equal to the number of joined tables (n_m). However, because both the subject and predicate for all triples are the same, we include the subject and predicate in the first triple and omit them from the rest of triples. We do this by separating objects by a comma as we mentioned before using *turtle* format. Thus, the format of the triple template (3.11) is $subject(\mathcal{T}_i \rightarrow \mathcal{T}_i) : \mathcal{T}_i\#hasNaryJoin\ object(\mathcal{T}_1 \rightarrow \mathcal{T}_1), object(\mathcal{T}_2 \rightarrow \mathcal{T}_2), \ldots, object(\mathcal{T}_m \rightarrow \mathcal{T}_m)$. The predicate $hasNaryJoin$ has the domain $:\mathcal{T}_i$ and the ranges $:\mathcal{T}_1$, $:\mathcal{T}_2$, ..., and $:\mathcal{T}_m$. OWL and $SPARQL$ cannot represent n-ary relations. To overcome this issue, we represent the n-ary relation by only one predicate that is named $hasNaryJoin$. It maintains the n-ary relation tightly-coupled by having all joined tables as its ranges.

The target part of an *Ontop* mapping rule for representing the n-ary join relation in $n - ary\ join\ table$ is generated by Lines 2 to 10 of Algorithm 3.2.6. The source query part is shown in (3.12). It is generated by calling the function *GenerateSqlQuery* in Line 11 of Algorithm 3.2.6. *GenerateSqlQuery* is shown in Algorithm E.0.1. Based on the value of the second parameter that is passed to the function by Algorithm 3.2.6 (i.e., "NARYJOIN"), the function *GenerateSqlQuery-ForNaryJoinRule* is called in Line 11 to generate a source query that is compatible with the *Ontop* mapping rule for an n-ary join relation on $n - ary\ join\ table$.

TARGET **:**

$$\underline{:\mathcal{T}_i/\mathcal{K}_{P_i^1} = \{\mathcal{K}_{P_i^1}\}\ [\ ;\ \mathcal{K}_{P_i^t} = \{\mathcal{K}_{P_i^t}\}\ ]^{t=2..n}}$$
$$\underline{:\mathcal{T}_i\#hasNaryJoin}$$
$$\underline{:\mathcal{T}_1/\mathcal{K}_{P_1^1} = \{\mathcal{K}_{P_1^1}\}\ [\ ;\ \mathcal{K}_{P_1^t} = \{\mathcal{K}_{P_1^t}\}\ ]^{t=2..n_1}}\ ,$$
$$\underline{:\mathcal{T}_2/\mathcal{K}_{P_2^1} = \{\mathcal{K}_{P_2^1}\}\ [\ ;\ \mathcal{K}_{P_2^t} = \{\mathcal{K}_{P_2^t}\}\ ]^{t=2..n_2}}\ ,$$
$$\ldots\ldots\ldots\ldots\ldots\ldots\ldots$$
$$\underline{:\mathcal{T}_m/\mathcal{K}_{P_m^1} = \{\mathcal{K}_{P_m^1}\}\ [\ ;\ \mathcal{K}_{P_m^t} = \{\mathcal{K}_{P_m^t}\}\ ]^{t=2..n_m}}\ . \tag{3.11}$$

SOURCE **:**

$$SELECT\ :\mathcal{T}_i.\mathcal{K}_{P_i^1}\ AS\ :\mathcal{T}_i\_\_\mathcal{K}_{P_i^1}\ [\ ,\ :\mathcal{T}_i.\mathcal{K}_{P_i^t}\ AS\ :\mathcal{T}_i\_\_\mathcal{K}_{P_i^t}\ ]^{t=2..n}\ ,$$
$$:\mathcal{T}_1.\mathcal{K}_{P_1^1}\ AS\ :\mathcal{T}_1\_\_\mathcal{K}_{P_1^1}\ [\ ,\ :\mathcal{T}_1.\mathcal{K}_{P_1^t}\ AS\ :\mathcal{T}_1\_\_\mathcal{K}_{P_1^t}\ ]^{t=2..n_1}\ ,$$
$$:\mathcal{T}_2.\mathcal{K}_{P_2^1}\ AS\ :\mathcal{T}_2\_\_\mathcal{K}_{P_2^1}\ [\ ,\ :\mathcal{T}_2.\mathcal{K}_{P_2^t}\ AS\ :\mathcal{T}_2\_\_\mathcal{K}_{P_2^t}\ ]^{t=2..n_2}\ ,$$
$$[\ ,\ :\mathcal{T}_x.\mathcal{K}_{P_x^1}\ AS\ :\mathcal{T}_x\_\_\mathcal{K}_{P_x^1}\ [\ ,\ :\mathcal{T}_x.\mathcal{K}_{P_x^t}\ AS\ :\mathcal{T}_x\_\_\mathcal{K}_{P_x^t}\ ]^{t=2..n_x}\ ]^{s=3..m}$$
$$FROM\ \mathcal{T}_i\ ,\ \mathcal{T}_1\ ,\ \mathcal{T}_2\ [\ ,\ \mathcal{T}_x\ ]^{s=3..m}$$
$$WHERE\ \mathcal{K}_{f_1^1}\ =\ \mathcal{K}_{P_1^1}\ [\ AND\ \mathcal{K}_{f_1^t}\ =\ \mathcal{K}_{P_1^t}\ ]^{t=2..n_1}$$
$$AND\ \mathcal{K}_{f_2^1}\ =\ \mathcal{K}_{P_2^1}\ [\ AND\ \mathcal{K}_{f_2^t}\ =\ \mathcal{K}_{P_2^t}\ ]^{t=2..n_2}$$
$$[\ AND\ \mathcal{K}_{f_x^1}\ =\ \mathcal{K}_{P_x^1}\ [\ AND\ \mathcal{K}_{f_x^t}\ =\ \mathcal{K}_{P_x^t}\ ]^{t=2..n_x}\ ]^{s=3..m} \tag{3.12}$$

n-ary Join Table Example. We explain Algorithm 3.2.6 through this example. Assume we have the tables *employee*, *component*, *product* and *assembly*, as shown in Listing 3.10. Table *assembly* represents an n-ary join table that connects the three tables *employee*, *component*, and *prdouct*. Both *employee* and *project* are independent tables. Thus, their *Ontop* mapping rules are extracted according to the *independent table* rule as shown in Listing 3.8.

```sql
CREATE TABLE employee
(
     empid int NOT NULL,
     empname varchar NOT NULL,
     PRIMARY KEY (empid)
);

CREATE TABLE component
(
     compid int NOT NULL,
     comptype varchar NOT NULL,
     compname varchar NOT NULL,
     PRIMARY KEY (compid)
);

CREATE TABLE product
(
     prodid varchar NOT NULL,
     prodtype varchar NOT NULL,
     prodname varchar NOT NULL,
     PRIMARY KEY (prodid)
);

CREATE TABLE assebmly
(
     empid int NOT NULL,
     compid int NOT NULL,
     prodid varchar NOT NULL,
     description varchar,
     PRIMARY KEY (empid, compid, prodid),
     CONSTRAINT assembly_employee_fk FOREIGN KEY (empid) REFERENCES employee (empid),
     CONSTRAINT assembly_component_fk FOREIGN KEY (compid) REFERENCES component (
     compid),
     CONSTRAINT assembly_product_fk FOREIGN KEY (prodid) REFERENCES product (prodid)
);
```

Listing 3.10: An example on a relational schema for Nary-Join relationship table rule.

First, we extract *Ontop* mapping rules for the four tables as shown in Listing 3.11. The *independent table* rule applies on the first three tables. Table *assembly* has three foreign keys that are composing the $3 - ary$ join of tables *employee*, *component*, and *prdouct*. Thus, the $n - ary$ *join* rule applies. However, because all foreign keys in *assembly* are part of the $3 - ary$ join relation, an *Ontop* mapping rule is extracted from *assembly* in the same way as in the *independent table*. Second, table *assembly* has a primary key that is composed of the foreign key fields *empid*, *compid*, and *prodid* that are referencing *employee*, *component*, and *prdouct*, respectively. Hence, an *Ontop* mapping rule that represents this $3 - ary$ relation is extracted as shown in Listing 3.12. The target part of the rule is according Template (3.11) and the source part is according to Template (3.12).

```
target      :Employee/empid={empid} a :Employee ;
            :Employee#empid {empid}^^xsd:integer ; :Employee#empname {empname} .
source      SELECT * FROM employee

target      :Component/compid={compid} a :Component ;
            :Component#compid {compid}^^xsd:integer ;
            :Component#comptype {comptype} ;
            :Component#compname {compname} .
source      SELECT * FROM component

target      :Product/prodid={prodid} a :Product ; :Product#prodid {prodid} ;
            :Product#prodtype {prodtype} ; :Product#prodname {prodname} .
source      SELECT * FROM product

target      :Assembly/empid={empid};compid={compid};prodid={prodid} a :Assembly ;
            :Assembly#empid {empid}^^xsd:integer ;
            :Assembly#compid {compid}^^xsd:integer ;
            :Assembly#prodid {prodid} ; :Assembly#description {description} .
source      SELECT * FROM assembly
```

Listing 3.11: Ontop Mapping rules for the four tables in Listing 3.10.

```
target        :Assembly/empid={assembly_empid};
                          compid={assembly_compid};
                          prodid={assembly_prodid}
                   :Assembly#hasNaryJoin
                          :Employee/empid={employee_empid} ,
                               :Component/compid={component_compid} ,
                                    :Product/prodid={product_prodid} .
source        SELECT   assembly.empid AS assembly_empid,
                       assembly.compid AS assembly_compid,
                       assembly.prodid AS assembly_prodid,
                       employee.empid AS employee_empid,
                       component.compid AS component_compid,
                       product.prodid AS  product_prodid
              FROM   assembly, employee, component, product
              WHERE    assembly.empid = employee.empid and
                       assembly.compid = component.compid and
                       assembly.prodid = product.prodid
```

Listing 3.12: Ontop Mapping rule for Nary-Join Table (ASSEMBLY) on Listing 3.10.

3.3 Extracting OWL Ontology from Ontop Mapping Rules

The process of extracting an equivalent OWL ontology for a relational schema from existing *Ontop* mappings is shown in Algorithm 3.3.1. The algorithm takes the extracted *Ontop* mappings $\mathcal{M}$ for the schema Σ and returns the extracted OWL ontology $\mathcal{W}$. It scans the *Ontop* mappings in $\mathcal{M}$. For each *Ontop* mapping rule, it checks the target templates; if the type of the template is a *class triple*, a new class is added to $\mathcal{W}$; if its type is an *object property triple*, a new object property is added to $\mathcal{W}$; and if it is a *data property triple*, a new data property is added to $\mathcal{W}$. In addition, the domain and range of each extracted object or data property are also extracted and added to $\mathcal{W}$. Thus, at the end of this process we have a complete OWL ontology with classes, properties, and relations. Real world examples on how

ALGORITHM 3.3.1: Extract OWL Ontology from Ontop mapping rules.

```
input  : $\Sigma$, $\mathcal{M}$
output: $\mathcal{W}$
1  foreach $\mu_i \in \mathcal{M}$ do
2      $target_i \leftarrow \mu_i.GetTarget()$ ;
3      foreach $triple \in target_i$ do
4          if $triple.type$ = "$ClassTriple$" then
5              $\mathcal{W} \leftarrow \mathcal{W} \cup \{C_j \leftarrow \texttt{CreateConcept}(triple)\}$;
6          else if $triple.type$ = "$ObjectPropertyTriple$" then
7              $OP_j \leftarrow \texttt{CreateObjectProperty}(triple)$;
8              $OP_j.domain \leftarrow GetDomain(triple)$;
9              $OP_j.range \leftarrow GetRange(triple)$;
10             $\mathcal{W} \leftarrow \mathcal{W} \cup OP_j$;
11         else if $triple.type$ = "$DataPropertyTriple$" then
12             $DP_j \leftarrow \texttt{CreateDataProperty}(triple)$;
13             $DP_j.domain \leftarrow GetDomain(triple)$;
14             $DP_j.range \leftarrow GetRange(triple)$;
15             $\mathcal{W} \leftarrow \mathcal{W} \cup DP_j$;
16 return $\mathcal{W}$;
```

to extract ontology elements from *Ontop* mappings are discussed in the following (listings 3.13, 3.15, and 3.17).

3.4 Implementation and Experiments

In this section, we show some examples for accessing relational data through SPARQL queries. The end users write their SPARQL queries against the extracted ontology. Note that users do not have any knowledge about the *Ontop* mappings that link the ontology to the underlying data source.

Example 6. Table 3.2 represents data instances for *dept* and *emp* in Listings 3.1 and 3.3, respectively. The example verifies the correctness of *Ontop* mappings for Independent, Dependent, and Recursive rules. For this purpose, we use the SPARQL query shown in Listing 3.14. It is based on the extracted ontology in Listing 3.13.

The optional part in the SPARQL query represents the case where the employee does not have a manager (to avoid a null result). The result of the SPARQL query is shown in Table 3.5.

Table 3.2: Data instance for the relational schema from Listings 3.1 and 3.3.

DEPT

deptno	dname	loc
10	Accounting	New York
20	Research	Dallas
30	Sales	Chicago
40	Operations	Boston

EMP

empno	empname	job	mgr	hiredate	sal	deptno
7839	King	President	Null	17-Nov-1981	5000	10
7566	Jones	Manager	7839	02-Apr-1981	2975	20
7788	Scott	Analyst	7566	09-Dec-1982	3000	20
7902	Ford	Analyst	7566	03-Dec-1981	3000	20
7369	Smith	Clerk	7902	17-Dec-1980	800	20
7782	Clark	Manager	7839	09-Jun-1981	2450	10
7934	Miller	Clerk	7782	23-Jan-1982	1300	10
7698	Blake	Manager	7839	01-May-1981	2850	30
7499	Allen	Salesman	7698	20-Feb-1981	1600	30
7521	Ward	Salesman	7698	22-Feb-1981	1250	30
7654	Martin	Salesman	7698	28-Sep-1981	1250	30
7900	James	Clerk	7698	03-Dec-1981	950	30
7876	Adams	Clerk	7788	12-Jan-1983	1100	20

```
PREFIX : <http://experiments.org/
:Dept rdf:type owl:Class .  :Emp rdf:type owl:Class .
:Dept#deptno rdf:type owl:DatatypeProperty .
:Dept#dname rdf:type owl:DatatypeProperty .
:Dept#loc rdf:type owl:DatatypeProperty .
:Emp#deptno rdf:type owl:DatatypeProperty .
:Emp#empno rdf:type owl:DatatypeProperty .
:Emp#empname rdf:type owl:DatatypeProperty .
:Emp#job rdf:type owl:DatatypeProperty .
:Emp#mgr rdf:type owl:DatatypeProperty .
:Emp#hiredate rdf:type owl:DatatypeProperty .
:Emp#sal rdf:type owl:DatatypeProperty .
:Emp#hasDEPT rdf:type owl:ObjectProperty .
:Dept#hasEMP rdf:type owl:ObjectProperty .
```

```
:Emp#hasEMP rdf:type owl:ObjectProperty .
```

Listing 3.13: The result of applying *independent*, *dependent*, and *recursive* rules of the extraction algorithm for Listings 3.1 and 3.3.

```
PREFIX : <http://experiments.org/>
PREFIX emp: <http://experiments.org/emp#>
PREFIX dept: <http://experiments.org/dept#>
SELECT ?e ?eNAME ?eJOB ?eMGR ?mNAME ?dDEPTNO ?dDNAME
WHERE {
    ?e a :Emp .
    ?e emp:empno ?eEMPNO .
    ?e emp:empname ?eNAME .
    ?e emp:job ?eJOB .
    OPTIONAL
    {
        ?e emp:hasEmp ?eMGR .
        ?eMGR a :Emp .
        ?eMGR emp:empname ?mNAME .
    }
    ?e emp:hasDept ?dDEPTNO .
    ?dDEPTNO a :Dept .
    ?dDEPTNO dept:dname ?dDNAME .
}
```

Listing 3.14: A SPARQL query example to access the relational schemas from Listings 3.1 and 3.3 through the extracted ontology (from Listing 3.13) and using Ontop mappings from Listings 3.2, 3.4, 3.5 and 3.6.

Example 7. Table 3.3 represents data instances for *employee − project*, *employee*, and *project* in Listings 3.7. The example verifies the correctness of *Ontop* mappings for binary join rule. For this purpose, we use the SPARQL query shown in Listing 3.16. It is based on the extracted ontology in Listing 3.15. The result of the SPARQL query is shown in Table 3.6.

Table 3.3: Data instance for the relational schema from Listing 3.7.

EMPLOYEE

employee_id	fname	lname
37	Fraces	Newton
1234	Donald	Newton

PROJECT

project_id	project_name
10	Online Market
20	Flight Booking

EMPLOYEE_PROJECT

emp_id	proj_id
37	10
1234	10
1234	20

```
PREFIX : <http://experiments.org/

:Employee rdf:type owl:Class .
:Employee#employee_id owl:DatatypeProperty .
:Employee#fname owl:DatatypeProperty .
:Employee#lname owl:DatatypeProperty .

:Project rdf:type owl:Class .
:Project#project_id owl:DatatypeProperty .
:Project#project_name owl:DatatypeProperty .

:Employee#hasProject owl:ObjectProperty .
:Project#hasEmployee owl:ObjectProperty .
```

Listing 3.15: The result of applying the Binary Join rule of the extraction algorithm from Listing 3.7.

```
PREFIX : <http://experiments.org/>
PREFIX emp: <http://experiments.org/employee#>
PREFIX proj: <http://experiments.org/project#>
SELECT * WHERE {
    ?e a :Employee .
    ?e emp:employee_id ?empid .
    ?e emp:fname ?empfirstname .
    ?e emp:lname ?emplastname .
    ?e emp:hasProject ?project .
    ?project proj:project_id ?projid .
    ?project proj:project_name ?projname
}
```

Listing 3.16: A SPARQL query example to access the relation schema from Listing 3.7 through the extracted ontology from Listing 3.15 and using Ontop mappings from Listings 3.8 and 3.9.

Example 8. Table 3.4 represents data instances for *employee*, *product*, *component*, and *assembly* in Listings 3.10. Through this example we want to verify the correctness of *Ontop* mappings for n-ary join rules. For this purpose, we use the SPARQL query shown in Listing 3.18. It is based on the extracted ontology in Listing 3.17. The result of the SPARQL query is shown in Table 3.7.

```
PREFIX : <http://experiments.org/

:Employee rdf:type owl:Class .
:Employee#empid rdf:type owl:DatatypeProperty .
:Employee#empname rdf:type owl:DatatypeProperty .

:Component rdf:type owl:Class .
:Component#compid rdf:type owl:DatatypeProperty .
:Component#compname rdf:type owl:DatatypeProperty .
:Component#comptype rdf:type owl:DatatypeProperty .

:Product rdf:type owl:Class .
:Product#prodid rdf:type owl:DatatypeProperty .
:Product#prodname rdf:type owl:DatatypeProperty .
:Product#prodtype rdf:type owl:DatatypeProperty .

:Assembly rdf:type owl:Class .
:Assembly#compid rdf:type owl:DatatypeProperty .
```

```
:Assembly#description rdf:type owl:DatatypeProperty .
:Assembly#empid rdf:type owl:DatatypeProperty .
:Assembly#prodid rdf:type owl:DatatypeProperty .

:Assembly#hasNaryJoin rdf:type owl:ObjectProperty .
```

Listing 3.17: The result of applying the n-ary Join rule of the extraction algorithm from Listing 3.10.

```
PREFIX : <http://experiments.org/>
PREFIX assembly: <http://experiments.org/assembly#>
PREFIX employee: <http://experiments.org/employee#>
PREFIX component: <http://experiments.org/component#>
PREFIX product: <http://experiments.org/product#>
PREFIX rdfs: <http://www.w3.org/2000/01/rdf-schema#>
SELECT ?assemblyId ?empName ?compType ?compName ?prodType ?prodName ?assemblyDesc
WHERE {
    ?assemblyId a :Assembly .   ?empId a :Employee .
    ?compId a :Component .   ?prodId a :Product .
    ?assemblyId assembly:hasNaryJoin ?empId .
    ?assemblyId assembly:hasNaryJoin ?compId .
    ?assemblyId assembly:hasNaryJoin ?prodId .
    ?empId employee:empname ?empName .
    ?compId component:comptype ?compType .
    ?compId component:compname ?compName .
    ?prodId product:prodtype ?prodType .
    ?prodId product:prodname ?prodName .
    ?assemblyId assembly:description ?assemblyDesc .
}
```

Listing 3.18: A SPARQL query example to access the relation schema from Listing 3.10 through the extracted ontology from Listing 3.17 and using Ontop mappings from Listings 3.11 and 3.12.

Table 3.4: Data instance for the relational schema from Listing 3.10.

EMPLOYEE

empid	empname
1	James Bond
2	John Smith

PRODUCT

prodid	prodtype	prodname
B22	Y22	Corvette
B33	X55	Camoro

COMPONENT

compid	comptype	compname
563	A33	Wheel
872	M16	Mirror
882	H55	Door hinge
883	H66	Trunk hinge
888	T53	Truck handle

ASSEMBLY

empid	compid	prodid	description
1	563	B22	assembled first
1	872	B22	assembled second
1	563	B33	assembled third
1	872	B33	assembled forth
2	563	B22	assembled fifth
2	882	B22	assembled sixth
2	888	B22	assembled seventh

Table 3.5: The SPARQL query's result from Listing 3.14.

e	eNAME	eJOB	eMGR	mNAME	dDEPTNO	dDNAME
:Emp/empno=7782	CLARK	MANAGER	:Emp/empno=7839	KING	:Dept/deptno=10	ACCOUNTING
:Emp/empno=7839	KING	PRESIDENT	null	null	:Dept/deptno=10	ACCOUNTING
:Emp/empno=7934	MILLER	CLERK	:Emp/empno=7782	CLARK	:Dept/deptno=10	ACCOUNTING
:Emp/empno=7369	SMITH	CLERK	:Emp/empno=7902	FORD	:Dept/deptno=20	RESEARCH
:Emp/empno=7566	JONES	MANAGER	:Emp/empno=7839	KING	:Dept/deptno=20	RESEARCH
:Emp/empno=7788	SCOTT	ANALYST	:Emp/empno=7566	JONES	:Dept/deptno=20	RESEARCH
:Emp/empno=7876	ADAMS	CLERK	:Emp/empno=7788	SCOTT	:Dept/deptno=20	RESEARCH
:Emp/empno=7902	FORD	ANALYST	:Emp/empno=7566	JONES	:Dept/deptno=20	RESEARCH
:Emp/empno=7499	ALLEN	SALESMAN	:Emp/empno=7698	BLAKE	:Dept/deptno=30	SALES
:Emp/empno=7521	WARD	SALESMAN	:Emp/empno=7698	BLAKE	:Dept/deptno=30	SALES
:Emp/empno=7654	MARTIN	SALESMAN	:Emp/empno=7698	BLAKE	:Dept/deptno=30	SALES
:Emp/empno=7698	BLAKE	MANAGER	:Emp/empno=7839	KING	:Dept/deptno=30	SALES
:Emp/empno=7900	JAMES	CLERK	:Emp/empno=7698	BLAKE	:Dept/deptno=30	SALES

Table 3.6: The SPARQL query's result from Listing 3.16.

e	empid	empfirstname	emplastname	project	projid	projname
employee/37	"37"^^xsd:integer	Fraces	Newton	project/10	"10"^^xsd:integer	Online Market
employee/1234	"1234"^^xsd:integer	Donald	Newton	project/10	"10"^^xsd:integer	Online Market
employee/1234	"1234"^^xsd:integer	Donald	Newton	project/20	"20"^^xsd:integer	Flight Booking

Table 3.7: The SPARQL query's result from Listing 3.18.

assemblyId	empName	compType	compName	prodType	prodName	assemblyDesc
assembly/empid=1;compid=563;prodid=B22	James Bond	A33	Wheel	Y22	Corvette	assembled first
assembly/empid=2;compid=563;prodid=B22	John Smith	A33	Wheel	Y22	Corvette	assembled fifth
assembly/empid=1;compid=563;prodid=B33	James Bond	A33	Wheel	X55	Camoro	assembled third
assembly/empid=1;compid=872;prodid=B22	James Bond	M16	Mirror	Y22	Corvette	assembled second
assembly/empid=1;compid=872;prodid=B33	James Bond	M16	Mirror	X55	Camoro	assembled forth
assembly/empid=2;compid=882;prodid=B22	John Smith	H55	Door hinge	Y22	Corvette	assembled sixth
assembly/empid=2;compid=888;prodid=B22	John Smith	T53	Truck handle	Y22	Corvette	assembled seventh

3.5 Related Work

There have been two prevalent models defined in the literature for representing ontologies on the Semantic Web: using OWL for ontology description, or through RDFS. A relational entity is usually mapped to either an OWL class or an RDFS class, and the attributes or columns in an entity are mapped to the defined class's properties. In OWL, there are two types of properties: data and object properties. A data property describes an attribute of a concept. It has the concept as its domain and a data type (i.e., integer, string, etc.) as its range. An object property describes a relationship between two ontological concepts. A general rule for extracting properties from a relational table is to extract a data property for each non-foreign key attribute and an object property for each foreign key reference. RDF statements are the data instances that conform to an ontology. After extracting an equivalent ontology to the relational schema, the relational instance can then extracted and transformed into RDF statements. Thus, we can distinguish two things in ontology-based data access. First, extracting an ontology (OWL or RDFS) that is equivalent to a relational schema. Second, extracting RDF statements that are equivalent to the schema instance and that conform to the extracted ontology.

A number of works in the literature exist for extracting an ontology from an existing relational schema. Most of these approaches share common rules for the extraction process. The most common rules that are repeated in the different approaches (e.g., in (Astrova, 2009), (Buccella et al., 2004), (Albarrak and Sibley, 2009), (Lubyte and Tessaris, 2009), (Sonia and Khan, 2008), (Cerbah, 2008), (Curino et al., 2009), (Alalwan et al., 2009), and (Tirmizi et al., 2008)) are as follows: default (or basic approach), binary relationship, n-ary relationship, hierarchy, and fragmentation rules (See (Spanos et al., 2012) for a comprehensive survey). A default rule is simply a basic and naive approach for extracting equivalent OWL concepts and properties from relations and their attributes. The basic approach is to map a relation to an OWL class, a non-foreign key attribute to an OWL data property, a foreign key attribute to an OWL object property, and a relation row to an individual of an OWL class. The first two rules of our proposed approach (i.e., independent and dependent rules) are similar to the basic approach. In addition, all the previously mentioned approaches have similar rules for the default and binary rules. (Albarrak and Sibley, 2009), (Cerbah, 2008), and (Curino et al., 2009) do not have a rule for representing the n-ary relationship. The hierarchy (or sub-class) rule is not considered by (Buccella et al., 2004), (Lubyte and Tessaris, 2009), and (Curino et al., 2009), whereas the fragmentation rule is not considered in all mentioned approaches, except in (Alalwan et al., 2009) . Hierarchy and fragmentation rules are very similar and one cannot distinguish between them unless he/she knows the intent of the schema designers or by mining the relational instance. The latter usually requires mining the data instance if enough data are available or using heuristic approaches, etc. Thus, most existing approaches misrepresent the hierarchy and fragmentation rules. In other words, they apply the hierarchy rule when in fact they should apply the fragmentation rule, and vice versa. In our approach, we decide to leave the discussion of the hierarchy and fragmentation rules for future research as these depend primarily on the availability of a relational instance with sufficient data. In addition, we leave the discussion of extracting equivalent OWL axioms for the relational constraints for future research.

Relational.OWL (de Laborda and Conrad, 2005) uses OWL itself to extract the semantics of the relational schema and represent it as an OWL/RDFS ontology. It then can represent the data instances of the relational schema as instances of the schema ontology. The primary advantage of this approach is that both schema and data changes can be automatically reversed on the schema ontology and its instance. However, a primary drawback of the Relational.OWL approach is that all the relational data should be transferred into RDF triples or OWL individuals before using SPARQL requests to access the data, which is an inefficient approach, especially when we have a huge data volume (Konstantinou et al., 2008). DB2OWL (Ghawi and Cullot, 2007) is a similar tool used to extract an OWL ontology from a relational schema. It only considers a few cases mentioned earlier, such as binary relation and sub-table relation. However, unlike our approach it does not consider recursive or n-ary relations. Furthermore, it does not discuss the case when a binary relation has other non-key attributes. It also uses an R2O mapping language (Barrasa et al., 2004) to automatically generate a mapping file that defines the relationship between the extracted ontology components and the relational schema. OntoAccess (Hert et al., 2010) focuses on the bidirectional access into the relational databases through SPARQL queries and SPARQL update languages. Thus, it is possible to update the underlying data sources through SPARQL updates in addition to querying them using SPARQL queries. However, it only supports basic relation to class and attribute to property mappings.

D2RQ (Bizer and Seaborne, 2004) is another widely used approach for ontology-based data access. It generates an RDFS ontology that is equivalent to a relational schema. It provides access to relational data through virtual RDF graphs that comply with the RDFS schema. However, it applies a simple approach (i.e., table to class and column to predicate) for extracting the mappings automatically. Furthermore, several studies show that the performance of its queries start to degrade when the size of the data is huge and it even stops responding in some cases. In addition, it does not support OWL ontologies.

Current approaches do not consider the evolution of both database schemas and ontologies due to the needs of end users and frequently changed applications.

However, we believe that approaches should adapt the current *Ontop* mappings to those changes, instead of re-extracting them from scratch. We are also planning to add a Managing module to the proposed approach. The managing module will be responsible for managing the extracted OWL ontology and the *Ontop* mapping rules between it and the relational schema. It will maintain both ontology and mapping rules consistency by frequently watching the relational schema and responding to any change. It will also update the ontology and the mappings rules after every change made on the relational schema when it is necessary.

Most current approaches are also interested only in the unidirectional access to the relational databases through Semantic Web ontologies. In other words, they only address the read-only access and do not consider write access (i.e., insert, update and delete). Research in the area of ontology-based data update has started to gain more popularity after the emergence of SPARQL Update language for updating RDF graphs (Seaborne et al., 2008). Furthermore, SPARQL update is recently sent to $W3C$ as a recommendation after several years of improvements (Passant et al., 2013). To date, only a few research works have used the SPARQL update language for the purpose of updating the relational databases through SPARQL queries (Spanos et al., 2012). The idea is to write SPARQL update requests according the SPARQL update language, transform them into equivalent SQL manipulation statements, and accordingly update the underlying relational databases. Examples are D2RQ/Update (Eisenberg and Kanza, 2012) that extends the D2RQ framework to write access to relational data and OntoAccess that provides both read and write access to the relational data (Hert et al., 2010). However, the previously mentioned works only covered basic SPARQL/Update statements to SQL statements. Moreover, the SPARQL Update language is still in the development process and it has not reached a mature level yet. The SPARQL Update language should be extended so it can support complex mapping SPARQL/Update to SQL statements. In future work, we plan also to extend the *Ontop* framework to support bidirectional Semantic Web based data access.

3.6 Conclusion and Future Work

We proposed an approach for extracting *Ontop* mappings and OWL ontology from a relational schema. The proposed approach defines an *Ontop* mapping rule's template for each type of relational entities. It covers the extraction rules for independent tables, dependent tables, recursive tables, binary join tables, and n-ary join tables. It also defines algorithms for automatically extracting *Ontop* mappings for a relational entity according the the defined templates. After all the *Ontop* mappings are extracted from the relational schema, the process of extracting an OWL ontology from those extracted mappings rules becomes a straightforward.

Our proposed approach can be extended to cover other type of relations, such as fragment entities, sub-entities (inheritance), enumerated attributes, and others. In addition, we will develop approaches for extracting equivalent ontological elements for database constraints (i.e., check, enum, null, not null, etc.) as discussed in the introduction.

Current approaches do not consider the evolution of both database schemas and ontologies due to the needs of end users and frequently changed applications. However, we believe that approaches should adapt the current *Ontop* mappings to those changes, instead of re-extracting them from scratch. We are also planning to add a Managing module to the proposed approach. The managing module will be responsible for managing the extracted OWL ontology and the *Ontop* mapping rules between it and the relational schema. It will maintain both ontology and mapping rules consistency by frequently watching the relational schema and responding to any change. It will also update the ontology and the mappings rules after every change made on the relational schema when it is necessary.

Most current approaches are also interested only in the unidirectional access to the relational databases through Semantic Web ontologies. In other words, they only address the read-only access and do not consider write access (i.e., insert, update and delete). Research in the area of ontology-based data update has started to gain more popularity after the emergence of SPARQL update language for updating RDF graphs (Seaborne et al., 2008). Furthermore, SPARQL update is recently sent to $W3C$ as a recommendation after several years of improvements (Passant et al., 2013). To date, only a few research works have used the SPARQL update language for the purpose of updating the relational databases through SPARQL queries (Spanos et al., 2012). The idea is to write SPARQL update requests according the SPARQL update language, transform them into equivalent SQL manipulation statements, and accordingly update the underlying relational databases. Examples are D2RQ/Update (Eisenberg and Kanza, 2012) that extends the D2RQ framework to write access to relational data and OntoAccess that provides both read and write access to the relational data (Hert et al., 2010). However, the previously mentioned works only covered basic SPARQL/Update statements to SQL statements. Moreover, the SPARQL update language is still in the development process and it has not reached a mature level yet. The SPARQL update language should be extended so it can support complex mapping SPARQL/Update to SQL statements. In future work, we plan also to extend the *Ontop* framework to support bidirectional semantic web based data access.

PART III: RESOLVING CONFLICTS EFFICIENTLY IN CDSS USING COMMUNITY FEEDBACKS

CHAPTER 4: Collaborative Data Sharing Systems and Conflicts

4.1 Preliminaries

A collaborative data sharing system facilitates users (usually in communities) to work together on a shared data repository to accomplish their (shared) tasks. Users of such a community can add, update, and query the shared repository (Gatterbauer et al., 2009) (please see (Overbeek et al., 2004; Buneman et al., 2006a; Bairoch et al., 2005; Tudor and Dvornich, 2001; Gouveia et al., 2004) for examples of some collaborative projects). While the shared database evolves over time and users extend it continuously, it may contain inconsistent data, as users may have different beliefs about which information is correct and which is not (Gatterbauer and Suciu, 2010). While a relational database management system (RDBMS) can be used to manage the shared data, RDMSs lack the ability to handle such conflicting data (Gatterbauer et al., 2009).

In most scientific communities (Ives et al., 2005; Taylor and Ives, 2006; Green et al., 2007; Ives et al., 2008; Kot and Koch, 2009), there is usually no consensus about the representation, correction, and authoritativeness of the shared data and corresponding sources (Ives et al., 2008). For example, in bioinformatics, various sub-communities exist where each focuses on a different aspect of the field (e.g., genes, proteins, diseases, organisms, etc.), and each manages its own schema and database instance. Still these sub-disciplines may have sharing links with their peer communities (e.g., a sharing link between genes and proteins sub-communities). A collaborative data sharing system thus needs to support these communities (and associated links), and provide data publishing, import, and reconciliation support for inconsistent data.

Traditional integration systems usually assume a global schema such that autonomous data sources are mapped to this global schema, and data inconsistencies are solved by applying conflict resolution strategies ((Naumann et al., 2006), (Bleiholder and Naumann, 2006), (Bilke et al., 2005), (Motro and Anokhin, 2006), (Bleiholder et al., 2007), and (Motro and Anokhin, 2004) are example systems). However, queries are only supported on the global schema and these systems do not support any kind of update exchange. To remedy this shortcoming, peer data management systems (Bernstein et al., 2002; Halevy et al., 2003) support disparate schemas, but are not flexible enough to support the propagation of updates between different schemas, and handling data inconsistency issues. In contrast, a collaborative data sharing system (CDSS) (Ives et al., 2005; Taylor and Ives, 2006; Green et al., 2007; Ives et al., 2008) allows groups of scientists that agree to share related data among them, to work on disparate schemas and database instances. Each group (or peer) can extend, curate, and revise its own database instance in a disconnected mode. At some later point, the peer may decide to publish the data updates publicly to other peers and/or get the updates from other peers. The reconciliation process in the CDSS engine (that works on top of the DBMS of each participant peer) is responsible for propagating updates and handling the disagreements between different participant peers. It publishes recent local data updates and imports non-local ones since the last reconciliation. The imported updates are filtered based on trust policies and priorities for the current peer. It then applies the non-conflicting and accepted updates on the local database instance of the reconciling peer. For the conflicting updates, it groups them into individual conflicting sets of updates. Each update of a set is assigned a *priority level* according to the trust policies of the reconciling peer. The reconciliation process then chooses from each set, the update with the highest priority to be applied on the local database instance, and rejects the rest. When it finds that many updates have the same highest preference or there is no assigned preferences for the updates in a set, it marks those updates as “deferred”. The deferred updates are not processed and not considered in future reconciliations until a user manually resolves the deferred conflicts.

4.1.1 Problem Description

The administrator of each peer in a CDSS is usually responsible for declaring and managing trust policies. While the administrator can be expected to define trust policies for a small number of participant peers, the same is not true for a large number of participants. In addition, assuming that a community of hundreds or thousands of members can authorize a user or a group of users to define trust policies for their community may not be plausible. Moreover, a CDSS does provide a semi-automatic conflict resolution approach by accepting the highest-priority conflicting updates, but it leaves for individual users the responsibility of resolving conflicts for the updates that are deferred. However, the assumption that individual users can decide how to resolve conflicting updates is not strong, as users of the community may have different beliefs and may agree or disagree with each other about which conflicting updates to accept and why (i.e., on which bases). Therefore, the challenge lies in providing a conflict resolution framework that requires minimal or no human intervention.

The remainder of this part is organized as follows. Section 4.2 serves as a brief introduction of the data integration problem and conflict resolution in conventional integration systems. Conflict resolution in community-based collaborative data sharing systems is presented in Section 4.3. Discussion and future directions are presented in Section 4.4. We then discuss the proposed approach for automated conflict resolution in a CDSS in Chapter 5.

4.2 Data Integration

A conventional integrative information system aims to combine heterogeneous (and possibly autonomous) data sources or schemas to provide users with a single unified (and usually reconciled) view of the data, which is known as a global or mediated schema (Hull, 1997; Ullman, 1997; Halevy, 2001). The heterogeneous data sources may have different data models, schemas, and data representations (Motro and Anokhin, 2006). The global schema provides a single representation for any real-world object that might have multiple representations in different data sources. In other words, when users submit queries against the global schema, they should not be aware about the multiple and heterogeneous data sources behind this global schema, and the query result should contain a consistent answer in respect to all the heterogeneous data sources. The most common integration scenario for integrating multiple and heterogeneous sources into a unified view is composed of three steps (Naumann et al., 2006). The three steps are schema matching and mapping, duplicate detection, and data fusion. Before the data integration process starts, we should have access to the remote data sources, which is currently solved by many technologies, like ODBC and JDBC connections, Web services, and many others. In the following, we provide a brief overview of the three steps of the data integration process.

4.2.1 Schema mapping

Assuming that the technical issues with connecting to the remote sources are solved, the first step in the data integration process is the resolution of schematic heterogeneity (Naumann et al., 2006). Schema mapping is an approach that is used to resolve the heterogeneity in data sources. It assumes that we are given two heterogeneous schemas, a source and a target (or a global schema), and the goal is to generate a set of correspondences between attributes of the source schema and the attributes of the global schema. Such correspondence determines how users' queries over the global schema are answered. The purpose of the mapping between source and global elements is to specify how to transform the data in a source element to a target element, such that the transformed data conforms to the global schema (Popa et al., 2002; Melnik et al., 2005).

We mention here two basic approaches that are used to establish such a mapping between source schemas and the global schema (Lenzerini, 2002): The first approach is called GAV (Global-As-View), which generates a global schema that conforms to the data sources (i.e., the global schema is just a view over the available data sources). The second approach is called LAV (Local-As-View), where the global schema is independent from the sources, and each source is represented as a view over the global schema. User queries are answered through the global schema, and users usually are not aware of the fact that the data are gathered from multiple heterogeneous data sources. A query over the global schema needs to be initially reformulated in terms of a set of queries over the heterogeneous data sources (Lenzerini, 2002).

An important addition to schema mapping are the schema matching techniques that initially and semi-automatically try to find a set of element correspondences between two schemas. (Rahm and Bernstein, 2001a) has classified schema matching techniques based on the type and level of information that each method can handle. The outcome of the schema mapping of two (or possibly more) heterogeneous schemas is that all entities from different schemas but represent the same thing are represented homogenously (i.e., a mapping is found between equivalent objects in both schemas).

Different schema matching techniques have been described in the literature. These techniques vary according to the type and level of the information that are used in the matching process. For example, the matching techniques can be schema-based, instance-based, attribute-based, constraint-based, etc. The Match operation is defined as "a function that takes two schemas as input and returns a mapping between elements of the two schemas that corresponds semantically to each other as output" (Rahm and Bernstein, 2001b,a).

4.2.2 Duplicate detection

The second step in the data integration process is duplicate detection (see (Elmagarmid et al., 2007) for a comprehensive survey about duplicate record detection). It aims to identify the different objects in the sources and thus find the multiple (and possibly inconsistent) representations of the same real-world objects (if any) (Bleiholder and Naumann, 2006). The outcome of the duplicate detection step is an addition of an ID field that is assigned for each real-world object. Objects that have the same ID are considered to be duplicates as they represent the same representation for a real-world object. Duplicate detection techniques can be classified into two groups: field matching techniques and duplicate record detection techniques (Elmagarmid et al., 2007).

Field matching techniques

Most of the field matching techniques that have been introduced in the literature compare the data at the attribute-level, focusing mainly on the comparison of string data (with less approaches defined to deal with numerical data). These techniques are classified into three groups: character-based, token-based, and phonetic-based similarity metrics (Elmagarmid et al., 2007). The character-based matching techniques are suitable for handling typographical errors. In contrast, the token-based matching techniques are suitable to find the duplication between two strings when the position of some words in the strings are fully misplaced, thereby character-based matching techniques are not suitable in this case. The last group of matching techniques measure the phonetic similarity between strings.

An example on a technique used to measure character-based similarity is *Levenshtein distance* (aka Edit distance) (Levenshtein, 1966). It measures the similarity between two strings by computing the minimum number of edit operations required to transfer one string to another, where the allowed operations on the first string are insertion of a single character into the string, deletion of a single character from the string, and replacing one character from the string with another character.

Atomic strings (Monge and Elkan, 1996) is an example on token-based similarity techniques. An atomic string is a sequence of alphanumeric characters delimited by punctuation characters. The algorithm considers two atomic strings are similar if they completely match or one is a prefix of the other. The similarity between two data elements is then computed by dividing the number of matched atomic strings in both data elements on the average of their total atomic strings.

Duplicate record detection techniques

Common duplicate record detection techniques can be classified into probabilistic, learning-based, distance-based, and rule-based models (Elmagarmid et al., 2007). The probabilistic models use the Bayesian theory to classify pair of records as matched or unmatched. Supervised learning techniques (Cochinwala et al., 2001; Tejada et al., 2002, 2001) assume the existing of a training set of pair records in which their label, whether matched or not matched, is already known. Any new unknown pair of records can then be classified according to the training data set. Rule-based techniques (Wang and Madnick, 1989; Lim et al.) depend on a set of defined rules to decide whether two records match or not, while distance-based approaches (Monge and Elkan, 1996, 1997; Dey et al., 1998) use a single or combination of field matching techniques to find if a pair of records are similar or not without the need for training data.

4.2.3 Data fusion

Data fusion is the last step of the integration process. It takes the output of the previous step as input and tries to fuse the duplicate representations of the same real-world object together into a single consistent representation after resolving such inconsistencies in the data (Naumann et al., 2006). Data fusion can be implemented using different approaches of conflict resolution strategies as we will see next (See (Bleiholder and Naumann, 2009) for a comprehensive survey about conflict classifications, strategies, and systems in heterogeneous sources).

Data conflicts

Data conflicts (or inconsistencies) issue comes after duplicate detection is done, when multiple representations of the same real-world objects are found. The conflict occurs when two or more matched tuples that represent the same object having inconsistency in some correspondent attributes' values. Data conflicts can be classified into data contradictions and data uncertainties. A contradiction is when two or more duplicate tuples have different non-null values for the same correspondent attributes. The most common reasons that lead to such conflict are typos, misspellings, outdated data values, or even when the different sources do not agree on the value. An uncertainty is when one of the duplicate tuples has a non-null value while the others have null values for the same correspondent attributes. The most common reasons that lead to such uncertainties are missing information or completely missing attribute in one tuple.

Conflict handling strategies

Conflict handling strategies in the literature are generally classified into three categories based on the procedures used to handle conflicting data (Bleiholder and Naumann, 2006): ignorance, avoidance, and resolution. In conflict ignorance strategies, an integration system usually does not need to be aware about data conflicts as they are ignored at all. *pass it on* and *consider all possibilities* are two examples of conflict ignorance strategies. The first one keeps all conflicting values and passes them to the user or application so they can decide what to do to solve these conflicts. The second one considers all possible combinations of attribute values and passes them to the user or application to let them decide which combination to choose (Burdick et al., 2005). Conflict avoidance strategies usually decide at the beginning whether to handle a conflict or not. When handling a conflict, they decide which values to choose even before looking at the conflicting values, though they do not resolve a conflict and even are not aware of it. Conflict avoidance strategies, in turns, can be classified into two classes based on whether taking into account metadata when deciding which value to choose: instance based and metadata based. *take the information* and *no gossiping* are two examples of instance based avoidance strategies. The first strategy is only suitable for data uncertainties, where unnecessary null values are ignored (Bleiholder and Naumann, 2006). The second one just takes the consistent data and leaves aside the inconsistent ones. An example of metadata based conflict avoidance is *trust your friends*. In this strategy, the user can have the option to prefer data from one source over data from other sources. TSIMMIS (Chawathe et al., 1994; Papakonstantinou et al., 1996; Garcia-Molina et al., 1997) and Hermes (Subrahmanian et al., 1995) are two systems that implement trust your friends strategy. On the other hand, conflict resolution strategies look at conflicting values and related metadata before deciding on how to resolve a conflict. They can be divided into two subcategories based on values chosen for resolution: deciding and mediating strategies. A deciding strategy chooses a value from the already present conflicting values, whereas a mediating strategy may choose a value that is not existing in conflicting values. *cry with the wolves* and *roll the dice* are two examples of instance-based, deciding strategies. The first one chooses the most common value among the conflicting ones, while the second chooses a random value among the conflicting ones. An example of a metadata-based, deciding strategy is *keep up to date* that chooses the most recent value. *Meet in the middle* is an example of a mediating strategy that tries to invent a value that is close to all conflicting values.

4.3 CDSSs Conflict Resolution Overview

Several studies have been recently introduced in the area of collaborative data sharing communities. Some of these studies describe conflict resolution approaches to deal with data inconsistency issues. BeliefDB (Gatterbauer et al., 2009) adopts an approach to resolve conflicts by using annotations to represent different beliefs of users in data sharing communities. Similarly, (Ives et al., 2005) and (Taylor and Ives, 2006; Green et al., 2007; Gatterbauer and Suciu, 2010) adopt trust mappings to resolve conflicts in community shared databases. (Pichler et al., 2010) adopts an approach for conflict resolution by collecting feedbacks from users of the local community. We summarize below the above mentioned works in more details.

4.3.1 BeliefDB

BeliefDB (Gatterbauer et al., 2009) is a new annotation-based database model. Annotations are usually realized as superimposed information that have been used recently for the purpose of explaining, correcting, or refuting the base data without actually modifying the data items (Maier and Delcambre, 1999). In other words, annotations are a kind of metatdata which are added to the existing data, usually without any underlying semantics (Srivastava and Velegrakis, 2007). Recently, annotations have gained popularity in the field of database community (Bhagwat et al., 2005; Buneman et al., 2006b, 2001; Chiticariu et al., 2005; Geerts and Van Den Bussche, 2007; Geerts et al., 2006).

The motivation of the BeliefDB model is to handle the conflicts that might arise in the shared data of scientific database applications. In this type of applications, a group of users or scientists have a shared repository and they all contribute to it by adding, updating, and revising operations. Relational database management systems (DBMSs) are used to manage such shared data, but they lack the ability to express and manage the conflicting facts in the database.

A belief database contains both base data in the form of tuples and belief statements that annotate these tuples. Users are enabled to annotate existing data or even exiting annotations by adding their own beliefs that might agree or disagree with the exiting facts. In other words, annotations should express the conflict between what users believe and what others believe. Thus, BeliefDB represents a set of belief worlds, such that each world belongs to a different user.

Moreover, a belief-aware query language is introduced to represent queries over a belief database. This query language can be used to retrieve facts that are believed or not believed by a particular user. It also can be used to query for the agreements or disagreements on particular facts between users. An algorithm is also described to translate belief database queries into equivalent relational SQL queries.

4.3.2 Trust mappings

A semi-automatic data conflict resolution for community shared databases based on trust mappings between users is proposed in (Gatterbauer and Suciu, 2010). In a community shared database, many users participate in a project and share the same data repository. They can add, update, or monitor the shared repository. For example, a group of scientists work together on a scientific project, etc. Each user can agree or disagree on any data value that is shared in the database. Users usually have trust relationships between each other in the community, such that a user can assign different trust weight to each user. The paper states that trust mapping is a relation between two users where one is willing to accept the other's data value. While the database is growing, it might contain conflicting information. Users can resolve conflicting data that comes from different trusted users by accepting the data value that comes from the most trusted user, by means of priorities. For example, a user X trusts data values that come from a user Y more than data values that come from a user Z. In such a shared database, each user is shown his own consistent version of the shared database based on his trust mappings and priorities with other users, such that a user can assign different trust weight to each user.

4.3.3 Uncertain databases

(Pichler et al., 2010) describes an approach that depends on users rating to handle inconsistent data in collaborative data sharing communities. It is similar to the work done in (Gatterbauer et al., 2009), as both assume a multi-versioned database model.

However, each user in (Gatterbauer et al., 2009) is shown a version of the database that is consistent with his/her own beliefs. Whereas in (Pichler et al., 2010), all users see a consistent version of the database that has the best rating. This work assumes that a group of users in collaboration, are working on a shared database. All updates done by users are stored in the shared database. For conflicting updates, all versions of those updates will be inserted into the database in parallel, resulting a multi-versioned database.

Users can update, query, and even rate the quality of updates (or data items). Each user, based on her own beliefs, can rate the quality of an update. The rating is usually weighted according to the reputation of the user who does the rating. Conflicting updates are usually various versions of the same tuple, sharing the same key, but having different values for non-key attributes. For each version of a single tuple (i.e., for each update), the rating of different users are collected, and the average rating for this version is computed. The reputation of a user who initiates the rated update can be then computed by comparing aggregate ratings of his updates to aggregate ratings of others. The computation of a user's reputation is incrementally, such that a new reputation value is computed for the user each time a new rating arrives. It can also be possible to compute the average rating of each version (or world) of the whole database.

For answering a query from a user, the average rating of each consistent version of the database is computed, and the best rated world is found. After that, a user query is answered according to this consistent version of the database. It is possible here that more than one world have the same best rating. In this case, a world with the most recent updates is preferred.

4.3.4 Youtopia

Youtopia (Kot and Koch, 2009) enables a community of users to manage collaboration and integration of relational data. Youtopia provides a notion for update exchange, similar to that of (Green et al., 2007), by allowing for changes to the data to propagate through a set of user-defined mappings, or in other words, tuple-generating dependencies (tgds). Youtopia borrows the concept of best-effort cooperation from the Web 2.0 and tries to satisfy it as much as possible. The best-effort means that the system should allow for any user to add their content on the Internet even if it is incomplete, as other users of the community may have the knowledge to complete this data later. The main layer in the architecture of the system is the Storage Manager. The goal of the storage manager layer is to provide the logical abstraction of the tables and views in the repository where the data resides. Users are responsible for managing (i.e., define, update, and delete of mappings when needed) the set of mapping rules that relates tables to each other, for the purpose of propagating any updates to the data.

Youtopia uses a backward (tgd) chase procedure (that extends the classical (tgd) chase procedure (Maier et al., 1979)) to propagate any changes to the data by chasing the affected mappings and doing the required modification, whether inserting, updating, or deleting some tuples. A violation of a tgd usually occurs after a user insert, delete, or update a particular tuple where the mappings between two relations are no longer satisfied. In backward chase procedure, Youtopia combines the classical chase with user intervention. A user can intervene, for example, when there is a need to delete some tuples that would handle the violation of a particular tgd rule. In addition, Youtopia can use forward and backward chase to correct any violation of a tgd rule. Youtopia differs from Orchestra in that it does not collect data provenance information.

4.3.5 Collaborative Data Sharing System (CDSS)

A CDSS is a new architecture introduced recently to support collaborative data sharing communities in general, and scientific communities specifically. The motivation behind the developing of this architecture is to enable scientific communities (e.g., bioinformatics) to share data among them. In such type of communities, usually there is no consensus about the representation, correction, and authoritative sources of shared data. In bioinformatics, for example, a group of heterogeneous peers might collaborate by sharing some related data among each other. You can consider peers as different sub-communities of bioinformatics field; say one interested in genes, another in proteins, and another in diseases, etc. Each peer has its own schema and its own database instance, but it might have associations with other peers. For example, there might be an association link between genes and proteins peers. According to the type of link, uni- or bi-direction, a peer needs to synchronize its version of shared data with other peers whom it has associations with. The synchronization can be accomplished by frequently exchanging updates among associated peers.

CDSS was first introduced in (Ives et al., 2005) as a new architectural model for collaborative data sharing. (Taylor and Ives, 2006) continues the effort to realize the CDSS by focusing on the propagation and reconciliation of updates between participant peers. Finally, (Green et al., 2007) completes on the previous works by introducing new methods to exchange peer's updates by using mappings and provenance information. We summarize below the work done in the above mentioned works in order.

4.3.6 Orchestra CDSS

Traditional integration systems follow a top-down approach by defining a global schema, where many heterogeneous sources are mapped to. This global schema is then targeted by user queries to get consistent answers. In addition, it does not support updates. However, in some scientific communities, the case is different, where the need is for bottom-up collaborative data sharing. In such communities, groups of scientists are working on different schemas and different data instances and agree to share related data among them. Each group has full control over its own version of the shared data usually by curating, revising, and extending this data. Each group, at some time, decides to publish the most recent updates to other groups and it gets, in turn, the recent updates from the others. It then applies the accepted updates and rejects the rest, keeps its database instance consistent and free of conflicts. This process is usually called the reconciliation.

Orchestra (Ives et al., 2008) is a prototype system that addresses the issues of collaborative data sharing for the certain type of communities mentioned above. In this system, a group of autonomous participants, each has its own local database, are collaborating together by sharing some related data. Each participant works on its local instance and later publishes the updates to others. Participants are connected together through direct relation. The direction of the relation is from the independent participant to the dependent one. The dependent participant usually needs to import the recent updates from participants whom have links with, and it applies these updates to its local instance taking in the consideration keeping its instance consistent. A participant in Orchestra usually operates in disconnected mode. While it is in disconnected mode, it can extend, curate, and modify its local instance, and later it decides to publish and/or reconcile. The reconciliation of a specific participant is the process of exchanging updates with the others and applying the incoming updates to the local instance after filtering the updates based on trust policies and priorities and rejecting the conflicting updates. The first step in the reconciliation is to import the recent updates from others since the last reconciliation. Second, determine which updates to accept based on the instance mappings, and rejecting the conflicting updates. Third, propagates to the participant relation only those updates that are both accepted and do not have conflicts. At last the reconciliation process records the updates made by the participant or accepted by it for future reconciliations.

4.3.7 Reconciliation

This paper (Taylor and Ives, 2006) is the first effort to realize the CDSS (Ives et al., 2005), a new architecture for supporting collaborative data sharing communities. The end goal of CDSS is to enable data sharing across disparate schemas and disparate data instances. This study assumes a group of autonomous participants or peers, sharing a single schema, and each one manages its own database instance. Data sharing links might exist between the different participants. Each participant is willing to share its data and updates with the others.

This study focuses on the propagation of updates among participants, which is the central problem in CDSS and it is usually called reconciliation. Participants make updates to their local database instances, and they later publish the updates upon their decisions. Each participant has acceptance rules that filter the update exchange based on the trust priority level for updates coming from the other participants. The job of the reconciliation process of a participant is to determine which updates are accepted and which are rejected. All updates that satisfies the acceptance rules and do not conflict with either the accepted updates or the reconciling participant's instance state are accepted. The priorities defined in acceptance rules are utilized

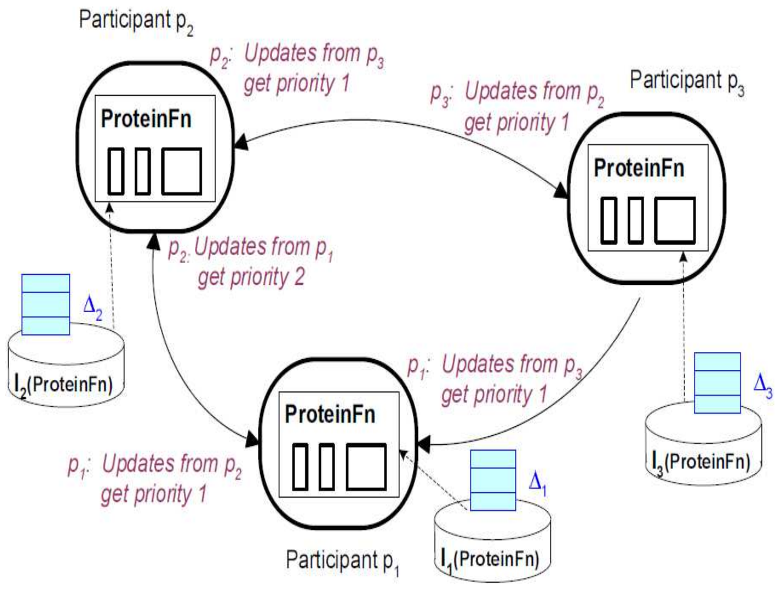

Participant p2
p2: Updates from p3 get priority 1
p3: Updates from p2 get priority 1
Participant p3
ProteinFn
ProteinFn
p2: Updates from p1 get priority 2
Δ2
I2(ProteinFn)
p1: Updates from p3 get priority 1
Δ3
ProteinFn
I3(ProteinFn)
p1: Updates from p2 get priority 1
Δ1
Participant p1
I1(ProteinFn)

Figure 4.1: Collaborative data sharing system with three bioinformatics data warehouse participants sharing data on protein functions (Taylor and Ives, 2006).

to determine which conflicting updates, if any, to accept. If the conflicting updates have the same priority or have no priorities assigned at all, then all these updates are marked as deferred, and they are not accepted until a user resolves the conflict manually. The reconciliation process also marks any future updates that conflict with unresolved updates as deferred, too.

Scenario Example: Consider a CDSS community of three participant peers (p_1, p_2, and p_3) that represent three bioinformatics warehouses as shown in Fig. 4.1 (example from (Taylor and Ives, 2006)). The three peers share a single relation *F(organism, protein, function)* for protein function, where the key of the relation is composed of the fields *organism* and *protein*. Peer p_1 accepts updates from both p_2 and p_3 with the same trust priority level. p_2 accepts updates from both p_1 and p_3, but it assigns a higher priority for updates that come from p_1. p_3 only accepts updates that come from p_2. We illustrate the reconciliation operation of this CDSS example as shown in Table 4.8.

Table 4.8: Reconciliation of F(organism, protein, function) (Taylor and Ives, 2006).

t	p_3	p_2	p_1
0	$I_3(F)\|0$={}	$I_2(F)\|0$ = {}	$I_1(F)\|0$ = {}
1	$T_{3:0}$:{+F(rat,prot1,cell-metab;3)} $T_{3:1}$:{F(rat,prot1,cell-metab $\rightarrow$ rat,prot1,immune;3)} <publish and reconcile> $I_3(F)\|1$:{(rat,prot1,immune)}		
2		$T_{2:0}$:{+F(mouse,prot2,immune;2)} $T_{2:1}$:{+F(rat,prot1,cell-resp;2)} <publish and reconcile> $I_2(F)\|2$:{(mouse,prot2,immune), (rat,prot1,cell-resp)}	
3	<reconcile> $I_3(F)\|3$:{(mouse,prot2,immune), (rat,prot1,immune)}		
4			<reconcile> $I_1(F)\|4$:{(mouse,prot2,immune)} DEFER: {$T_{3:1}$, $T_{2:1}$}

In the beginning, we assume that the instance of relation F at each participant peer p_i, denoted by $I_i(F)|0$, is empty (i.e., at time 0). At time 1, p_3 conducts two transactions $T_{3:0}$ and $T_{3:1}$. It then decides to publish and reconcile its own state (to check if other peers made any changes). Since the other two participant peers have not yet published any updates, p_3's instance, after the reconciliation operation is completed, $I_3(F)|1$ denotes the result (the second transaction is only a modification to the first one). At time 2, p_2 conducts two transactions $T_{2:0}$ and $T_{2:1}$. It then publishes and reconciles its own state. Note that the resulting instance $I_2(F)|2$ of p_2 contains only its own updates. Although there is a recently published update by p_3, which is trusted by it, p_2 does not accept p_3's published update because it conflicts with its own updates. At time 3, p_3 reconciles again. It accepts the transaction $T_{2:0}$ that is published by p_2 and rejects p_2's second update $T_{2:1}$ because it conflicts with its own state. At time 4, p_1 reconciles. It gives the same priority for transactions of p_2 and p_3. Thus, it accepts the non-conflicting transaction $T_{2:0}$, and it defers the conflicting transactions $T_{2:1}$ and $T_{3:1}$.

4.4 Discussion and Future Directions

In recent years, few collaborative data sharing applications, which support data exchange among multiple and heterogeneous schemas, have emerged. As mentioned earlier, in these applications, users usually organize themselves in groups or communities, such that each community focuses on a specific, and probably distinctive domain. While each community manages its own schema, it can have sharing links with other communities. Sharing links enable different communities to exchange data between them in a managed and restrictive manner, that usually depends on predefined trust relationships and priorities. Examples applications can be found in scientific communities, academic communities (e.g., DBLP, ACM), blog communities, etc. (Ives et al., 2005). However, the exchanged data between different communities may be inconsistent and produce variable results. Thus, there is a need for efficient techniques to handle such inconsistency in data (and its integration).

Approaches for the problem of inconsistent data have been described in detail in the context of conventional integrations systems. For instance, (Naumann et al., 2006; Bleiholder and Naumann, 2006; Bilke et al., 2005; Motro and Anokhin, 2006; Bleiholder et al., 2007; Motro and Anokhin, 2004) described different approaches for conflict resolution while integrating heterogeneous database sources. However, all conventional integration systems usually aim to map heterogenous data sources to a unified global schema. Users can then submit their queries to this global schema without being aware of the involved heterogeneity. In addition, these systems are usually built on top of the Relational DBMS, which does not support conflicts in the data to be represented in the database. Thus, these integration systems are not sufficient to support the collaboration needs for users of scientific communities.

Approaches for handling conflicts in community shared databases, based on the concept of multi-versioned databases, are described in (Gatterbauer et al., 2009; Pichler et al., 2010; Gatterbauer and Suciu, 2010). In (Gatterbauer et al., 2009), a BeliefDB system enables users to annotate existing data or even existing annotations, by adding their own beliefs that may agree or disagree with existing data or annotations. (Gatterbauer and Suciu, 2010) describes an automatic conflict resolution technique based on trust mappings between users. To resolve the conflict, a user only accepts a data value that comes from the most trusted user. Thus, each user is shown his own consistent version of the shared database (based on his trust mappings and priorities with other users). In essence, a network of prior trust mappings between users have to be defined or known (Taylor and Ives, 2006). (Pichler et al., 2010) handles inconsistent data by allowing users to rate data. Updates done by users are stored in a shared, uncertain database, where all versions of conflicting updates are inserted into the database in parallel. The work done in (Pichler et al., 2010) is similar to that of (Gatterbauer et al., 2009; Gatterbauer and Suciu, 2010), in that all apply a multi-versioned database model to resolve conflicts. However, each user in (Gatterbauer et al., 2009; Gatterbauer and Suciu, 2010) sees his own consistent version of the shared database based on his own beliefs or trust mappings. In contrast, all users in (Pichler et al., 2010) see the most consistent version of the database which has the best rating. The shortcoming of all the above approaches that try to solve the problem of inconsistency in data by implementing a multi-versioned database, is that they only support the collaboration needs of a single community of users (and usually suitable for public communities of users). Thus, they do not support the collaboration needs for the sort of communities mentioned earlier.

In contrast, a CDSS (Ives et al., 2005; Taylor and Ives, 2006; Green et al., 2007; Ives et al., 2008) allows groups of scientists to work on disparate schemas and database instances, while each group can have sharing links with other groups. The reconciliation process in the CDSS engine (that works on top of the DBMS of each participant peer) is responsible for propagating updates and handling the disagreements between different participant peers. It publishes recent local data updates and imports non-local ones since the last reconciliation. The imported updates are filtered based on trust policies and priorities for the current peer. It then applies the non-conflicting and accepted updates on the local database instance of the reconciling peer. For the conflicting updates, it groups them into individual conflicting groups of updates. Each update of a group is assigned a *priority level* according to the trust policies of the reconciling peer. The reconciliation process then chooses from each group, the update with the highest priority to be applied on the local database instance, and rejects the rest. When it finds that many updates have the same highest preference or there is no assigned preferences for the updates in a group, it marks these updates as "deferred". The deferred updates are not processed and not considered in future reconciliations until a user resolves the deferred conflicts manually. The administrator of each peer in a CDSS is usually responsible for declaring and managing trust policies. While the administrator can successfully manage to define trust policies for a few number of participant peers, this task is not easy for a huge number of participants. In addition, assuming that a community of hundreds or thousands of members can authorize a user or a group of users to define trust policies for their community is usually not possible. Moreover, a CDSS does provide a semi-automatic conflict resolution approach by accepting the highest-priority conflicting updates, but it leaves for individual users the responsibility of resolving conflicts for the updates that are deferred. However, the assumption that individual users can decide how to resolve conflicting updates is not strong, as users of the community may have different beliefs and may agree or disagree with each other about which conflicting updates to accept and why (i.e., on which bases).

Table 4.9: Comparison between community-based systems and models .

	Heterogeneous data sources	Relational data support	Mediated schema	Update exchange	Conflict resolution support	Automatic conflict resolution	Object's global ID assumption	Trust mappings	Priorities	Update dependency	Community-based collaboration	Multiple communities support	Implementation
Orchestra (Ives et al., 2008)	X	X		X	X			X	X	X	X	X	X
Youtopia (Kot and Koch, 2009)	X	X		X				X			X		X
BeliefDB (Gatterbauer et al., 2009)	X	X			X		X				X		
DCUUTM (Gatterbauer and Suciu, 2010)	X	X			X		X	X	X		X		
UDBinCDM (Pichler et al., 2010)	X				X	X	X				X		
HumMer (Bilke et al., 2005)	X	X	X		X								X
Fusionplex (Motro and Anokhin, 2006)	X	X	X		X		X						X
FuSem (Bleiholder et al., 2007)	X	X	X		X								X
MBCR (Motro and Anokhin, 2004)	X	X	X		X		X		X				

We conclude that the conventional integration systems cannot support the collaboration needs for scientific communities. Recent collaborative sharing approaches in this regard usually manage one community of users and support a single schema, with disparate instances. However, they do not support collaboration needs for multiple distinctive communities. In addition, they are usually more suitable for public users who organize themselves in communities, where each community is specialized in a specific domain (e.g., sport, games, etc.). On the other hand, CDSS can fully support the collaboration needs between different communities that usually have disparate schemas and instances, but they may have some relations in common that require sharing links and data exchange. Table 4.9 shows a comparison between the different approaches discussed earlier. An X in the table cell means that the requirement stated in the column's header is supported by the approach or system stated in the row's header. We see that there is still a room to improve the CDSS. We list below some of the directions for future research works to improve the CDSS:

- Automatic Conflict Resolution:

  We believe a fully automated approach to resolve conflicts that may arise due to the propagation of updates among related peers in a CDSS is a prime requirement. Automation can be achieved by resolving conflicts in the deferred set (of a CDSS's reconciling peer) by collecting feedbacks about the quality of the conflicting updates from the local community (i.e., local users) and remote peers. However, the use of feedbacks should be limited. For instance, deploying the community feedback for the purpose of resolving conflicts between the updates of conflict groups in the deferred set of a local peer. This can enable each participant peer to maintain a relational and consistent database instance, where conflicts between data are not allowed due to the restrictions of the relational DBMS.

- Automatic Trust Mappings and Policy Definitions:

  We can extend the above work to utilize community feedbacks not only to resolve conflicts for the updates in the deferred set, but also to deploy community feedbacks for the purpose of automatically defining trust policies for the local peer, thereby omitting the role of the administrator in defining trust policies.

CHAPTER 5: Automated Conflict Resolution in CDSS

In this section, we discuss our approach for resolving conflicts in the set of conflict groups of updates that are added to the deferred set of a CDSS's participant peer during its reconciliation operation. Fig. 5.1 shows the general architecture of a CDSS participant peer using the proposed approach. Before further discussion, we need to define the key entities/players of the CDSS: (i) *Provider Peer* is the entity that shares its data updates with other peers in the CDSS. (ii) *Consumer/Reconciling Peer* is the entity that receives (possibly conflicting) updates on the same data from multiple providers. (iii) *Remote/Rater Peer* is the entity that helps the consumer in the reonciliation process by providing ratings about the provider. (iv) Multiple *Users* (which may be human) are registered with one peer in a mutually exclusive manner. In the proposed approach, after the reconciliation operation of the consumer adds a new conflict group to the deferred set, the following steps are taken:

1. The reconciliation operation inquires other remote peers (i.e., remote raters) about their past experiences with the provider peers that have conflicting updates in this conflict group. The following sub-steps are then taken to compute the remote assessed reputation of each provider:

    (a) After receiving all replies from remote raters, the credibility values of responding raters are (re)computed based on the majority rating and the aggregation of the previously, computed remotely assessed reputations of this provider.

    (b) Reported ratings provided by remote raters are then weighted according to the new credibility values. The credibility value of a remote rater represents to what degree the reconciling peer trusts the rating value reported by the remote rater.

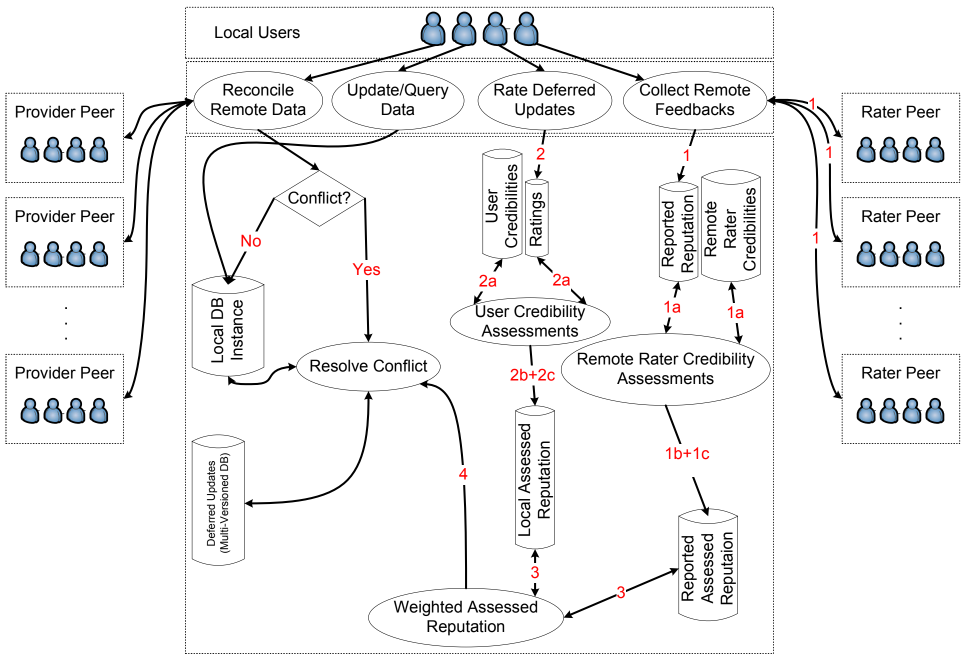


Figure 5.1: Proposed CDSS architecture

(c) Weighted reported reputation values are then aggregated for each update in the conflict group. This aggregated value represents the remotely assessed reputation of a particular provider peer as viewed by the reconciling peer.

2. The reconciliation operation informs local users of the reconciling peer to rate updates in this conflict group. Whenever this conflict group is rated by a number of users more than a predefined threshold, then it is marked as closed. Local users are thence not allowed to rate this closed conflict group or change their previous rating. The following sub-steps are then taken to compute the local assessed reputation of each provider peer:

(a) Whenever a conflict group is marked as closed, then for each provider peer that has an update in this conflict group, the credibility values of users that rate the updates of this provider peer, are (re)computed based on the majority rating and the aggregation of the previously computed locally assessed reputations of this provider.

   (b) The reported ratings provided by users are then weighted according to the new credibility values. The credibility value for a user represents to what degree the reconciling peer trusts the provided rating for the update of a particular provider peer.

   (c) Weighted reported ratings are then aggregated for each update in the conflict group. This aggregated value represents the locally assessed reputation of a particular provider as viewed by the reconciling peer.

3. The assessed reputation of each provider peer that is involved in the closed conflict group is computed by weighting both computed remotely and locally assessed reputations of this provider peer. The weights that are given for both computed values depend on the reconciling peer's administrator. The administrator may assign the local reputation of a provider higher weight than the remote reputation of a provider, or vice-versa.

4. Finally, the update which is imported from the provider peer with the highest assessed reputation value is applied to the reconciling peer's instance (making sure it does not violate its integrity constraints).

In the following, we describe in details how to compute both remote and local reputations of a provider peer. We assume a CDSS, where a group of autonomous peers share a single schema, and each one manages its own database instance. Every relation in the database has a key, and a tuple is an entry in the database identified by a key. Disagreement on the non-key values of a tuple leads to several versions of this tuple. Table 5.10 lists the definition of symbols used henceforth.

Table 5.10: Definition of symbols.

Symbol	Definition
P	Set of CDSS's participant peers $\{p_1, \ldots ,p_n\}$.
Σ	Schema that represents the relations in the system.
$I_i(\Sigma)$	Local database instance controlled by a peer p_i.
t	Reconciliation time counter.
p_i	Peer who is reconciling.
p_j	Remote peer.
G_c	Particular conflict group in the deferred set of p_i.
$g_{c:j}$	Particular conflicting update of G_c that is imported from remote peer p_j.
t_{G_c}	Closing Time of the rating process for an unresolved conflict group G_c.
p_i^x	Local user x of p_i who participates in the rating process.
σ_i	Threshold of % of raters for p_i to close the rating on updates of G_c.
h_x	Last h non-neutral rating by p_i^x to p_j's from already resolved conflict groups.
γ	Smoothing factor in the $interval[0, 1]$ for determining the weights of recent ratings.
$\mathcal{MR}$	Value of the majority rating.
$\mathcal{MR}_\Delta$	Change in credibility due to the majority rating.
$\overline{RRPP}$	Aggregation value of previously k assessed reputations of a particular peer.
$\overline{RRPP}_\Delta$	Effect on credibility due to agreement or disagreement with $\overline{RRPP}$.
Φ	Credibility adjustment normalizing factor.
Ψ	Amount of change in credibility.
ρ	Pessimism factor.
$f(\varphi)$	Aggregation function.

5.1 Remote Reputation of a Provider Peer (RRPP)

When a new conflict group is added to the deferred set of a consumer peer, it needs to resolve the conflict by choosing a single update from the group, and reject others. This decision is based on the feedbacks collected from both, other remote CDSS peers, and the local user community (that forms the consumer peer). In this section, we provide details on feedbacks collection from remote peers, while we discuss the feedbacks collected from the local user community in the next section.

In the proposed system, each CDSS participant peer records its perception of the reputation of the provider peer(s). This perception is called the *personal evaluation* of a provider peer in the consumer's view. In this study, we assume that a consumer peer computes this personal evaluation every time it needs to resolve a conflict for any conflict group added to its deferred set and only for provider peers that have their updates in this particular conflict group. Let p_j be a provider peer and p_x be a rater peer. p_x maintains $Rep(p_j, p_x)$ that represents its personal evaluation of p_j's reputation score. Other peers may differ or concur with p_x's observation of p_j. A consumer peer p_i that inquires about the reputation of a given provider peer p_j from rater peers may get various differing personal evaluations or feedbacks. Thus, to get a correct assessment of p_j, all the collected feedbacks about p_j need to be aggregated. The aggregation of all feedbacks collected from remote raters to derive a single reputation value ($RRPP$) represents p_j's remote assessed reputation as viewed by p_i. Consumer peers may employee different aggregation techniques. Formally, the remote assessed reputation $RRPP(p_j, p_i)$ of a provider peer p_j as viewed by a consumer peer p_i is defined as:

$$RRPP(p_j, p_i) = f(\varphi)_{x \in L}(Rep(p_j, p_x)) \tag{5.1}$$

where L denotes the set of rater peers which have interacted with p_j in the past and are willing to share their personal evaluations of p_j with p_i, $Rep(p_j, p_x)$ is the last personal evaluation of p_j as viewed by p_x, and $f(\varphi)$ represents the aggregation function, which can be simply the average of all feedbacks, or it can be a more complex process that considers a number of factors.

A major drawback of feedback-only based systems is that all ratings are assumed to be honest and unbiased. A provider peer that usually produces high quality updates may get incorrect or false ratings from different evaluators due to several malicious motives. In order to deal with this issue, a reputation management system should weigh the ratings of highly credible raters more than raters with low credibilities (Delgado and Ishii, 1999). In our approach, the reputation score of the provider peer is calculated according to the credibility scores of the rater peers. The credibility score of a rater peer p_x assigned by a consumer peer p_i determines to what degree p_i trusts the reputation value assigned by this rater to a provider peer p_j. Taking into consideration the credibility factor, the $RRPP$ of p_j is calculated as a weighted average according to the credibilities of the rater peers. Thus, the Equation (5.1) becomes:

$$RRPP(p_j, p_i) = \frac{\sum_{x=1}^{L}(Rep(p_j, p_x) * \mathcal{C}_{p_x})}{\sum_{x=1}^{L}\mathcal{C}_{p_x}} \tag{5.2}$$

where $\mathcal{C}_{p_x}$ is the credibility of p_x as viewed by p_i. The credibility of a rater peer lies in the interval [0,1] with 0 identifying a dishonest rater and 1 an honest one. The overall rater credibility assessment process follows.

Evaluating Rater Credibility: To minimize the effects of *unfair* or inconsistent ratings we screen the ratings based on their deviations from the majority opinion (similar to other works in (Buchegger and Boudec, 2004), (Whitby et al., 2005), (Walsh and Sirer, 2005), (Weng et al., 2005), etc). The basic idea is that if the reported rating agrees with the majority opinion, the rater's credibility is increased, and decreased otherwise. However, unlike previous models, we do not simply disregard/discard the rating if it disagrees with the majority opinion but consider the fact that the rating's inconsistency may be the result of an actual experience. Hence, only the credibility of the rater is changed, but the rating is still considered. We use a data clustering technique to define the majority opinion by grouping similar feedback ratings together. We use the k-mean clustering algorithm (Macqueen, 1967) on all current reported ratings to create the clusters. The most densely populated cluster is then labelled as the "majority cluster" and the centroid of the majority cluster is taken as the majority rating (denoted $\mathcal{MR}$). To obtain a better measure of the dispersion of ratings, we calculate the Euclidean distance between the majority rating ($\mathcal{MR}$) and each reported rating (R). The resulting value is then normalized using the standard deviation (σ) in all the reported ratings. The normalization equation (to assess the change in credibility due to majority rating), denoted by $\mathcal{MR}_\Delta$ is then defined as:

$$\mathcal{MR}_\Delta = \begin{cases} 1 - \frac{\sqrt{\sum_{k=1}^{n}(\mathcal{MR} - R_k)^2}}{\sigma} & \text{if } \sqrt{\sum_{k=1}^{n}(\mathcal{MR} - R_k)^2} < \sigma; \\ 1 - \frac{\sigma}{\sqrt{\sum_{k=1}^{n}(\mathcal{MR} - R_k)^2}} & \textit{otherwise.} \end{cases} \tag{5.3}$$

Note that $\mathcal{MR}_\Delta$ does not denote the rater's credibility (or the weight), but only defines the effect on credibility due to agreement/disagreement with the majority rating. How this effect is applied will be discussed shortly. There may be cases in which the majority of raters collude to provide an incorrect rating for a particular provider peer. Moreover, the outlier raters (ones not belonging to the majority cluster) may be the ones who are first to experience the deviant behavior of the providers. Thus, a majority rating scheme "alone" is not sufficient to accurately measure the reputation of a provider peer.

We supplement the majority rating scheme by adjusting the credibility of a rater based on the past behavior of a provider as well. The historical information provides an estimate of the *trustworthiness* of the raters (Sonnek and Weissman, 2005) (Whitby et al., 2004). The trustworthiness of a provider peer is computed by looking at the "last assessed reputation value" (for a provider peer p_j), the present majority rating for p_j, and the rater peer's corresponding provided rating. We define a credible rater as one which has performed consistently, accurately, and has proven to be useful (in terms of ratings provided) over a period of time.

We believe that under controlled situations, a consumer peer's perception of a provider peer's reputation should not deviate much, but stay consistent over time. We assume the interactions take place at time t and the consumer peer already has record of the previously assessed $RRPP$, then:

$$\overline{RRPP} = f(\varphi)_{t-1}^{t-k} RRPP(p_j, p_i)^t \tag{5.4}$$

where $RRPP(p_j, p_i)$ is the assessed $RRPP$ of a provider peer p_j by a consumer peer p_i for each time instance t, $f(\varphi)$ is the aggregation function and k is the time duration defined by each consumer peer. It can vary from one time instance to the complete past reputation record of p_j. Note that $\overline{RRPP}$ is not the "personal evaluation" of either the rater peer or the consumer peer but is the average of the "remote assessed reputation" calculated by a consumer peer at the previous time instance(s). If a provider behavior does not change much from the previous time instances, then $\overline{RRPP}$ and the present reported rating R should be somewhat similar. Thus, the effect on credibility due to agreement or disagreement with the aggregation of the last k assessed $RRPP$ values (denoted $\overline{RRPP_\Delta}$) is defined in a similar manner as in Equation (5.3):

$$\overline{RRPP_\Delta} = \begin{cases} 1 - \dfrac{\sqrt{\sum_{k=1}^{n}(\overline{RRPP} - R_k)^2}}{\sigma} & \text{if } \sqrt{\displaystyle\sum_{k=1}^{n}(\overline{RRPP} - R_k)^2} < \sigma; \\ 1 - \dfrac{\sigma}{\sqrt{\sum_{k=1}^{n}(\overline{RRPP} - R_k)^2}} & otherwise. \end{cases} \tag{5.5}$$

In real-time situations it is difficult to determine the different factors that cause a change in the state of a provider peer. A rater peer may rate the same provider peer differently without any malicious motive. Thus, the credibility of a rater peer may change in a number of ways, depending on the values of R, $\mathcal{MR}_\Delta$, and $\overline{RRPP_\Delta}$. The general formula is:

$$\mathcal{C}_{p_x} = \mathcal{C}_{p_x} \pm \Phi * \Psi \tag{5.6}$$

where Φ is the credibility adjustment normalizing factor, while Ψ represents amount of change in credibility due to the equivalence or difference of R with $\mathcal{MR}$ and $\overline{RRPP}$. The signs $\pm$ indicate that either $+$ or $-$ can be used, i.e., the increment or decrement in the credibility depends on the situation. These situations are described in detail in the upcoming discussion.

We place more emphasis on the ratings received in the current time instance than the past ones, similar to previous works as (Buchegger and Le Boudec, 2004) (Whitby et al., 2004). Thus, equivalence or difference of R with $\mathcal{MR}$ takes a precedence over that of R with $\overline{RRPP}$. This can be seen from Equation (5.6), where the $+$ sign with Φ indicates $R \simeq \mathcal{MR}$ while $-$ sign with Φ means that $R \neq \mathcal{MR}$. Φ is defined as:

$$\Phi = \mathcal{C}_{p_x} * (1 - |R_x - \mathcal{MR}|) \tag{5.7}$$

Equation (5.7) states that the value of the normalizing factor Φ depends on the credibility of the rater and the absolute difference between the rater's current feedback and the majority rating calculated. Multiplying by the rater's credibility allows the honest raters to have greater influence over the ratings aggregation process and dishonest raters to lose their credibility quickly in case of a false or malicious rating. The different values of Ψ are described next.

Adjusting Rater Credibilities: Ψ is made up of $\mathcal{MR}_\Delta$ and/or $\overline{RRPP_\Delta}$, and a "pessimism factor" (ρ), which is used to normalize the change factor (for rater credibility). The exact value of ρ is left at the discretion of the consumer peer, with the exception that its minimum value should be 2. The lower the value of ρ, the more optimistic is the consumer peer and higher value of ρ are suitable for pessimistic consumers (this value is inverted in Equations (5.10 and 5.11)). We define a pessimistic consumer as one that does not trust the raters easily and reduces their credibility drastically on each false feedback. Moreover, honest rater's reputations are increased at a high rate, meaning that such consumers make friends easily. On the other hand, optimistic consumers tend to "forgive" dishonest feedbacks over short periods (dishonesty over long periods is still punished), and it is difficult to attain high reputation quickly. Only prolonged honesty can guarantee a high credibility in this case. R, $\mathcal{MR}$, and $\overline{RRPP}$ can be related to each other in one of four ways, and each condition specifies how $\mathcal{MR}_\Delta$ and $\overline{RRPP_\Delta}$ are used in the model. Note that the normalizing factor (ρ in our case) is common among all the four conditions. The difference is in the different 'amounts', that are based on equalities or inequalities among R, $\mathcal{MR}$, and $\overline{RRPP}$. In the following, we provide an explanation of each and show how the credibilities are updated in our proposed model using different values for Ψ.

Case 1. The reported reputation value is similar to both the majority rating and the aggregation of the previously computed $RRPP$ values (i.e., $R \simeq \mathcal{MR} \simeq \overline{RRPP}$). The equality $\mathcal{MR} \simeq \overline{RRPP}$ suggests that majority of the raters believe that the quality of updates imported from a provider peer p_j has not changed. The rater peer's credibility is thus updated as:

$$\mathcal{C}_{p_x} = \mathcal{C}_{p_x} + \Phi * \left(\frac{|\mathcal{MR}_\Delta + \overline{RRPP_\Delta}|}{\rho}\right) \tag{5.8}$$

Equation (5.8) states that since all variables are equal, the credibility is incremented. We will see in the following that in the current case, the factor multiplied to Φ is the largest (due to the variable equalities).

Case 2. The individual reported reputation rating is similar to the majority rating but differs from the previously assessed reputation, i.e. ($R \simeq \mathcal{MR}$) and ($R \neq \overline{RRPP}$). In this case, the change in the reputation rating could be due to either of the following. First, the rater peer may be colluding with other raters to increase or decrease the reputation of a provider peer. Second, the quality of updates imported from the provider peer may have actually changed since $\overline{RRPP}$ was last calculated. The rater peer's credibility is updated as:

$$\mathcal{C}_{p_x} = \mathcal{C}_{p_x} + \Phi * \left(\frac{\mathcal{MR}_\Delta}{\rho}\right) \tag{5.9}$$

Equation (5.9) states that since $R \simeq \mathcal{MR}$, the credibility is incremented, but the factor $R \neq \overline{RRPP}$ limits the incremental value to $(\frac{\mathcal{MR}_\Delta}{\rho})$ (not as big as the previous case).

Case 3. The individual reported reputation value is similar to the aggregation of the previously assessed $RRPP$ values but differs from the majority rating, i.e. $(R \neq \mathcal{MR})$ and $(R \simeq \overline{RRPP})$. The individual reported reputation value may differ due to either of the following. First, p_x may be providing a rating score that is out-dated. In other words, p_x may not have the latest score. Second, p_x may be providing a "false" negative/positive rating for a provider peer. The third possibility is that p_x has the correct rating, while other rater peers contributing to $\mathcal{MR}$ may be colluding to increase/decrease the provider peers reputation. None of these three options should be overlooked. Thus, the rater peer's credibility is updated as:

$$\mathcal{C}_{p_x} = \mathcal{C}_{p_x} - \Phi * \left(\frac{\overline{RRPP_\Delta}}{\rho}\right) \tag{5.10}$$

Equation (5.10) states that since $R \neq \mathcal{MR}$, the credibility is decremented, but here the value that is subtracted from the previous credibility is adjusted to $(\frac{\overline{RRPP_\Delta}}{\rho})$.

Case 4. The individual reported reputation value is not similar to both the majority rating and the calculated aggregation of assessed $RRPP$ values, i.e. $(R \neq \mathcal{MR})$ and $(R \neq \overline{RRPP})$. p_x may differ from the majority rating and the past aggregation of $RRPP$ values due to either of the following. First, p_x may be the first one to experience the provider peer's new behavior. Second, p_x may not know the actual quality of the provider peer's imported updates. Third, p_x may be lying to increase/decrease the provider peer's reputation. In this case, the rater peer's credibility is updated as:

$$\mathcal{C}_{p_x} = \mathcal{C}_{p_x} - \Phi * \left(\frac{|\mathcal{MR}_\Delta + \overline{RRPP_\Delta}|}{\rho}\right) \tag{5.11}$$

Equation (5.11) states that the inequality of all factors means that rater peer's credibility is decremented, where the decremented value is the combination of both the effects $\mathcal{MR}_\Delta$ and $\overline{RRPP_\Delta}$.

5.2 Local Reputation of a Provider Peer (LRPP)

In our proposed solution, users can rate deferred updates according to their own beliefs about which update is the most correct.

5.2.1 Rating updates

The reconciliation operation in a consumer peer p_i notifies local users when a new conflict group of updates (G_c) is inserted into the deferred set $Deferred(p_i)$. It also specifies the *closing time* (t_{G_c}) of the rating process for this unresolved conflict group. Local users of p_i rate the updates of unresolved G_c in $Deferred(p_i)$. A user x (p_i^x) of p_i assigns a probabilistic rating ($r_{i:x,j}$) in the $interval[0, 1]$ to each update $g_{c:j}$ of a provider peer p_j in G_c, where 0 identifies the rater's extreme disbelief and 1 identifies the rater's extreme belief in an update. Moreover, a user can assign a neutral rating (-1) to an update to express his lack of opinion about this particular update. A trigger is fired to inform the reconciliation operation when a voting period of unresolved G_c is ended. The reconciliation operation then checks whether this G_c is rated by a number of users exceeding a predefined *percentage of the total number of local users* (σ_i). If the number of users who rate this G_c exceeds σ_i, the reconciliation operation marks this G_c as "closed" and users cannot rate this G_c anymore. Otherwise, the reconciliation operation extends the rating period of this particular G_c (to attain the threshold).

5.2.2 Computing the $LRPP$ value

We adopt the same technique introduced in Section 5.1 to compute the $LRPP$ value. Each participant peer records the past computed $LRPP$ values for each provider peer it works with. We also assume that a consumer peer computes a new $LRPP$ value every time it needs to resolve a conflict for any conflict group added to its deferred set and only for provider peers which they have their updates in this particular conflict group G_c. Then, $Rep(p_j, p_i^x)$ represents the rating assigned by a local consumer's user p_i^x to the update of provider p_j in G_c. Formally, the $LRPP$ of a provider peer p_j as viewed by a consumer peer p_i, computed post closing a conflict group G_c, is defined as:

$$LRPP(p_j, p_i) = \frac{\sum_{x=1}^{L}(Rep(p_j, p_i^x) * \mathcal{C}_{p_i^x})}{\sum_{x=1}^{L}\mathcal{C}_{p_i^x}} \tag{5.12}$$

where L denotes the set of local users who have rated p_j's update in G_c, $Rep(p_j, p_i^x)$ is the rating of p_j, and $\mathcal{C}_{p_i^x}$ is the credibility of a local user p_i^x as viewed by p_i. This Equation is the same as Equation (5.2). The only difference is that we here aggregate the summation of ratings given by local users, for the purpose of computing the $LRPP$ value for a particular provider peer. The credibility of a local user assigned by a parent peer p_i determines to what degree p_i trusts the ratings assigned by a local user to a provider peer p_j. As mentioned earlier, we follow the same approach discussed previously to compute the credibility of local users. We do not provide the details here to avoid redundancy as only minor changes are required. The only modification to Equations 5.1 through 5.11 is using ratings assigned by local users of a reconciling peer to provider peers' updates in a closed conflict group. Notice that $\overline{LRPP}$ is computed by an equation similar to $\overline{RRPP}$ (as in Equation (5.4)). However, $\overline{LRPP}$ represents the aggregation of the past $LRPP$ computed by p_i for p_j, assuming that p_i keeps records of the previously computed $LRPP$.

5.3 Illustrative Example

In this section, we provide a comprehensive example to illustrate the proposed approach. Let us consider a CDSS community of three participant peers (p_1, p_2, and p_3) that represents three bioinformatics warehouses (example adapted from (Taylor and Ives, 2006)). The three peers share a single relation *F(organism, protein, function)* for protein function, where the key of the relation is composed of the fields *organism* and *protein*. Peer p_1 accepts updates from both p_2 and p_3 with the same trust priority level. p_2 accepts updates from both p_1 and p_3, but it assigns a higher priority for updates that come from p_1. p_3 only accepts updates that come from p_2. For the purpose of the illustration, we also assume that there are 10 other participant peers (p_4 through p_{13}). In this example, we assign different roles for the participant peers. We consider peers p_2 and p_3 as provider peers for the rest of peers, peer p_1 as a consumer peer who imports updates from the provider peers and needs to reconcile its own instances. The remaining peers (p_4 through p_{13}) play the role of raters which are assumed to have interacted with the provider peers in the past and are willing to share their experiences with other consumer peers. Similar to (Taylor and Ives, 2006), we illustrate the reconciliation operation of this CDSS example as shown in Table 5.11, taking into consideration our proposed modification for the system.

In the beginning (i.e., at time 0), we assume that the instance of relation F at each participant peer p_i, denoted by $I_i(F)|0$, is empty. At time 1, p_3 conducts two transactions $T_{3:0}$ and $T_{3:1}$. It then decides to publish and reconcile its own state (to check if other peers made any changes). Since the other two participant peers have not yet published any updates, p_3's instance, after the reconciliation operation is complete; $I_3(F)|1$ denotes the result (the second transaction is only a modification to the first one). At time 2, p_2 conducts two transactions $T_{2:0}$ and $T_{2:1}$. It then publishes

Table 5.11: Reconciliation of F(organism, protein, function).

Time	p_3	p_2	p_1
0	$I_3(F)\|0 = \{\}$	$I_2(F)\|0 = \{\}$	$I_1(F)\|0 = \{\}$
1	$T_{3:0}$: {+F(rat, prot1, cell-metab; 3)} $T_{3:1}$: {F(rat, prot1, cell-metab $\rightarrow$ rat, prot1, immune; 3)} <publish and reconcile> $I_3(F)\|1$: {(rat, prot1, immune)}		
2		$T_{2:0}$: {+F(mouse, prot2, immune; 2)} $T_{2:1}$: {+F(rat, prot1, cell-resp; 2)} <publish and reconcile> $I_2(F)\|2$: {(mouse, prot2, immune), (rat, prot1, cell-resp)}	
3	<reconcile> $I_3(F)\|3$: {(mouse, prot2, immune), (rat, prot1, immune)}		
4			<reconcile> $I_1(F)\|4$: {(mouse, prot2, immune)} DEFER: {$T_{3:1}$, $T_{2:1}$ }
5	$T_{3:2}$: {+F(cat, prot3, cell-metab; 3)} <publish and reconcile> $I_3(F)\|5$: {(cat, prot3, cell-metab), (mouse, prot2, immune), (rat, prot1, immune)}		
6			<reconcile> $I_1(F)\|6$: {(rat, prot1, immune), (cat, prot3, cell-metab), (mouse, prot2, immune)}

and reconciles its own state. Note that the resulting instance $I_2(F)|2$ of p_2 contains only its own updates. Although there is a recently published update by p_3, which is trusted by it, p_2 does not accept p_3's published update because it conflicts with its own updates. At time 3, p_3 reconciles again. It accepts the transaction $T_{2:0}$ that is published by p_2 and rejects p_2's second update $T_{2:1}$ because it conflicts with its own state. At time 4, p_1 reconciles. It gives the same priority for transactions of p_2 and p_3. Thus, it accepts the non-conflicting transaction $T_{2:0}$, and it defers both the conflicting transactions $T_{2:1}$ and $T_{3:1}$.

p_1's reconciliation operation forms a conflict group G_1 (shown in Table 5.12) that includes both deferred transactions that are added to the deferred set of p_1 during the reconciliation. p_1 first inquires other remote peers about their trust placed in the provider peers that have conflicting updates in G_1. Second, it notifies its local users that a new conflict group is added to $Deferred(p_1)$, so they can start rating updates in this particular conflict group. The result of these two steps is the computing of $RRPP$ and $LRPP$ values for each provider peer that has an update in G_1 (p_2 and p_3 in this case). p_1 then computes the assessed trust for each provider peer who has update in G_1 by weighting the values of $RRPP$ and $LRPP$ according to its pre-defined preferences. Next, we provide the details of these steps.

Table 5.12: The deferred set of peer p_1.

G_c	Txn	p_1^1	p_1^2	p_1^3	p_1^4	p_1^5	p_1^6	p_1^7	p_1^8	p_1^9	p_1^{10}	Status	σ_i
G_1	$T_{3:1}$	0.95	0.65	1.00	0.60	0.97	1.00	0.95	0.90	0.95	1.00	Closed	100%
	$T_{2:1}$	0.45	0.80	0.45	0.75	0.40	0.40	0.45	0.40	0.45	0.43		

5.3.1 Computing the *RRPP*

We assume here that the local peer p_1 maintains a table of all the previously assessed reputation values of provider peers that it interacts with. For instance, the last 10

$RRPP$ values previously computed by p_1 for provider peers p_2 and p_3 are {0.58, 0.55, 0.56, 0.62, 0.60, 0.63, 0.59, 0.51, 0.53, 0.55} and {0.95, 1.00, 0.94, 0.89, 0.90, 0.94, 0.85, 0.87, 0.96, 0.92} respectively. Similarly, as mentioned earlier, p_1 maintains a credibility value for each rater peer that responds to its request for any p_j's rating.

After a new conflict group G_1 is added to the deferred set of p_1, assume that p_1 gets back responses from rater peers p_4, p_5, ..., p_{13}. The received responses (in-order) for p_2 are {0.70, 0.65, 0.50, 0.46, 0.52, 0.67, 0.55, 0.43, 0.47, 0.90}, and for p_3 are {0.98, 0.88, 0.93, 0.96, 0.99, 0.91, 0.90, 0.89, 0.95, 0.45 }. Using this information, p_1's reconciliation operation performs the following series of steps for each provider peer in G_1:

1. p_1 computes the values of $\mathcal{MR}$, $\mathcal{MR}_\Delta$, $\overline{RRPP}$, and $\overline{RRPP_\Delta}$ factors for each provider peer in G_1. The computed values for p_2 are (0.57, 0.59, 0.67, .67) and for p_3 are (0.92, 0.88, 0.68, 0.67), respectively.

2. p_1 computes the new credibility values for each rater peer, as shown in Table 5.13, who has provided their ratings for p_2. Then, it takes the new computed credibility values as an input to compute the new credibility values for consumer raters who provides their ratings to p_3, as shown in Table (5.14), assuming that each consumer rater has provided his rating for all provider peers that appear in the conflict group F_1. We provide more details about the computations done in Tables 5.13 (and 5.14) in the following:

    (a) The first row of Table 5.13, titled ($\mathcal{C}_{p_x}(old)$), shows the current credibility values for rater peers (p_4, p_5, ..., p_{13}).

    (b) In the second row of Table 5.13, the values of Φ variable are shown after Equation (5.7) is applied.

Table 5.13: Computing p_2's $RRPP$ and the new credibility values for remote raters who respond to the inquiry regarding the reputation of the provider peer p_2.

p_2	Remote Rater Peers									
Factor	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}
$C_r(x)_{old}$	0.95	0.97	0.89	0.88	0.97	0.90	0.95	0.93	0.94	0.95
Φ	0.84	0.91	0.81	0.77	0.91	0.82	0.92	0.79	0.83	0.65
$R \simeq \mathcal{MR}$	0.12	0.07	0.09	0.13	0.06	0.09	0.03	0.16	0.12	0.32
$\mathcal{MR} \simeq \overline{RRPP}$	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01	0.01
$R \simeq \overline{RRPP}$	0.13	0.08	0.07	0.11	0.05	0.10	0.02	0.14	0.10	0.33
$Case(1-4)$	1	1	1	1	1	1	1	1	1	4
Ψ	0.01	0.01	0.01	0.01	0.00	0.01	0.00	0.02	0.01	0.10
$C_r(x)_{new}$	0.96	0.97	0.90	0.89	0.97	0.91	0.95	0.95	0.95	0.88
R_w	0.67	0.63	0.45	0.41	0.51	0.61	0.52	0.41	0.45	0.79

(c) The rows (3-5) show the equalities between the factor pairs ($R \simeq \mathcal{MR}$), ($\mathcal{MR} \simeq \overline{RRPP}$), and ($R \simeq \overline{RRPP}$), for each consumer rater. Here, we assume that the two compared factors are equal if the amount of difference between them is equal or less than 0.20. Otherwise, they are considered not to be equal. If we look at Table 5.13, we see that all pairs are considered equal, except for the consumer rater p_{13}. For those raters who have ($R \simeq \mathcal{MR} \simeq \overline{RRPP}$), Case (1) conditions are met, and thus we apply Equation (5.8) for computing the new credibility values. For p_{13}, we have ($R \neq \mathcal{MR}$) and ($R \neq \overline{RRPP}$). Thus, Case (4) is met, and we apply Equation (5.11) for computing the new credibility value. Since the reported rating value by p_{13} is not similar to both the majority opinion and the aggregation of the previously computed $RRPP$ values of provider peer p_2, p_{13} is penalized (by decreasing its credibility and giving a less weight for its reported rating).

(d) The rows (6-8) of Table 5.13 show the matched case, the value of Ψ, and the new computed credibility value ($\mathcal{C}_{p_x}(new)$), for each rater.

(e) The last row, titled R_w, shows the weightage of reputation values received from the different raters.

(f) Based on the last two rows of Table 5.13, p_1's reconciling operation computes the $RRPP$ for provider peer p_2 ($RRPP(p_2, p_1) = 0.58$) by applying Equation (5.2).

Table 5.14 values are obtained in the same manner as defined above, and the $RRPP$ for p_3 is computed as $RRPP(p_3, p_1) = 0.89$ by applying Equation (5.2). Note that the new credibility values computed in Table 5.13 are used as inputs to compute the new credibility values for consumer raters who provided their reputation values for provider peer p_3. Again, credibilities of all consumer raters are altered according Case (1), except for consumer rater p_{13} where its credibility is altered according Case (4).

Table 5.14: Computing p_3's $RRPP$ and the new credibility values for remote raters who respond to the inquiry regarding the reputation of the provider peer p_3.

p_3	Remote Raters									
Factor	p_4	p_5	p_6	p_7	p_8	p_9	p_{10}	p_{11}	p_{12}	p_{13}
$C_r(x)_{old}$	0.96	0.97	0.90	0.89	0.97	0.91	0.95	0.95	0.95	0.88
Φ	0.87	0.97	0.85	0.82	0.87	0.88	0.94	0.94	0.89	0.50
$V \simeq \mathcal{MR}$	0.10	0.00	0.05	0.08	0.11	0.03	0.02	0.01	0.07	0.43
$\mathcal{MR} \simeq \overline{RRPP}$	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04	0.04
$V \simeq \overline{RRPP}$	0.06	0.04	0.01	0.04	0.07	0.01	0.02	0.03	0.03	0.47
$Case(1-4)$	1	1	1	1	1	1	1	1	1	4
Ψ	0.01	0.00	0.00	0.00	0.01	0.00	0.00	0.00	0.00	0.21
$C_r(x)_{new}$	0.97	0.98	0.90	0.89	0.98	0.91	0.95	0.95	0.95	0.78
V_w	0.95	0.86	0.83	0.86	0.97	0.83	0.86	0.84	0.90	0.35

5.3.2 Computing the $LRPP$

Note that the local peer p_1 (the reconciling peer in our running example) maintains a table of all previously assessed $LRPP$ values of provider peers that it interacts with. For instance, the last 5 $LRPP$ values for p_2 and p_3 are $\{0.41, 0.43, 0.58, 0.52, 0.38\}$ and $\{0.90, 0.89, 0.89, 0.94, 0.90\}$ respectively. Similarly, it maintains a credibility value for each local user (remember each peer is composed of n users) that has provided reputation ratings regarding different conflicts in the past. The credibility values change according to the new assessed $LRPP$ values of provider peers computed by the local peer p_1. Let us also assume here that all 10 users of p_1 (denoted p_1^1, p_1^2,..., p_1^{10}) have participated in the rating of all the updates in conflict group, and the rating process is considered to be closed, as illustrated in Table 5.12. When this requirement is met, p_1's reconciliation operation marks the conflict group G_1 as closed to inform users to stop giving new ratings to updates of this conflict group. After G_1 is marked as closed, $p1$'s reconciliation operation performs the same steps as in computing the $RRPP$ value above for each provider peer in G_1. We omit the $LRPP$ computation steps (and associated tabular results) here to avoid redundancy as they are very similar to the above mentioned steps that we follow to compute the $RRPP$ value. Instead, we only summarize the outcome of these steps as follows: (i) p_1 computes the values of $\mathcal{MR}$, $\mathcal{MR}_\Delta$, $\overline{LRPP}$, and $\overline{LRPP_\Delta}$ factors for each provider peer in G_1. The computed values for p_2 are (0.47, 0.50, 0.68, .67) and for p_3 are (0.90, 0.90, 0.67, 0.67), respectively. (ii) Based on the local user credibilities, and reported ratings p_1's reconciling operation computes the $LRPP$ for provider peers p_2 ($LRPP(p_2, p_1) = 0.48$) and p_3 ($LRPP(p_3, p_1) = 0.91$), respectively.

5.3.3 Conflict Resolution

After the conflict group $G1$ is closed, and $RRPP$ and $LRPP$ values are computed for each provider peer in G_1, p_1's reconciliation operation computes the assessed reputations of provider peers p_2 and p_3. The assessed reputation of a provider peer is computed by weighing the $RRPP$ and $LRPP$ values. As mentioned earlier, the administrator of the reconciling peer p_1 is responsible for defining the appropriate weightages. For our example, let us assume that the weight given for the $RRPP$ is 40% and for the $LRPP$ is 60%. Thus, the assessed reputation of p_2 will be

$(Rep(p_2, p_1) = 0.58 * 40\% + 0.48 * 60\% = 0.52)$, and the assessed reputation of p_3 will be $(Rep(p_3, p_1) = 0.89 * 40\% + 0.91 * 60\% = 0.90)$. Since p_3 has the higher reputation value, the transaction $T_{3:1}$ of p_3 is considered in the next reconciliation operation, and applied to the local instance of peer p_1 as it does not violate its local state or does not conflict with other accepted transactions during the reconciliation. However, the transaction $T_{2:1}$ of p_2 is rejected, and it is not considered in the next reconciliations.

In continuation of the scenario as illustrated in Table 4.8, at time 5, p_3 applies a new transaction $T_{3:2}$. It then decides to publish and reconciles its own state, and it ends with the instance $I_3(F)|5$. At time 6, p_1 decides to reconcile. It ends up with applying the transaction $T_{3:1}$, resulting from the ratings on the updates of conflict group G_1, to its local instance. It also accepts and applies the new published transaction $T_{3:2}$ of p_3. Hence, p_1 ends up with $I_1(F)|6$.

5.4 Implementation Model and Results

In this section, we illustrate the implementation details of the proposed approach using the above mentioned scenario. We modeled the different entities (as defined at the beginning of this section) as Java-based simulator to see how the algorithms perform with large number of conflicts and different qualities of providers and raters. The experiments are conducted in a closed environment, where we can capture the actual behavior of providers and raters. The validity of the proposed approach can thus be measured by observing the difference between the actual behavior of the providers and raters, and their computed reputation values and credibilities, respectively. The provider CDSS updates are created in a semi-automated manner, to follow one of five classes of providers (details in Section 5.4.2). Similarly, the percentages of honest and dishonest raters are changed to see their impact on the proposed approach.

5.4.1 One Consumer and Multiple Providers

In the first set of experiments, we developed a CDSS with three participant peers. p_1 is the reconciling peer, whereas p_2 and p_3 are the provider peers. p_1 has 100 local users. The provider peers are initially assigned degrees of quality or behavior randomly on a scale of $[0, 1]$ where 0 denotes the lowest quality and 1 the highest. For p_2, the value lies between 0.1 and 0.7, and between 0.7 and 1.0 for p_3. We further divided the One consumer and Multiple providers case into two sets of experiments. In the first set, 80% of users are high quality users (i.e., they provide accurate rating values in the range $[0.8, 1]$, and 20% of them are low quality users (i.e., they provide poor rating values in the range $[0.1, 0.4]$). In the second set, we keep the quality level of rating for both groups of users the same as in the first set, but we only increase the percentage of dishonest raters (to 50%) and decrease the percentage of honest ones (to 50%). At the beginning of the simulation, we assume that all local users of the reconciling peer have credibility of 1. Each time during the simulation, p_2 and p_3 generate identical tuples (i.e., tuples that have the same key but differ in values of the non-key attributes) and then publish their updates. When p_1 reconciles (i.e., imports the newly published updates from both p_2 and p_3), a conflict is found in the pair of updates with the same key but imported from different providers. The conflict is resolved by either accepting the update of p_2 or p_3, according to the weighted ratings of users. The simulation ends when p_1 resolves the conflict numbered 3600.

Fig. 5.2 shows the results for the above mentioned experiment sets. For conciseness, the average of 10 rounds of experiments is shown. In the first set (denoted

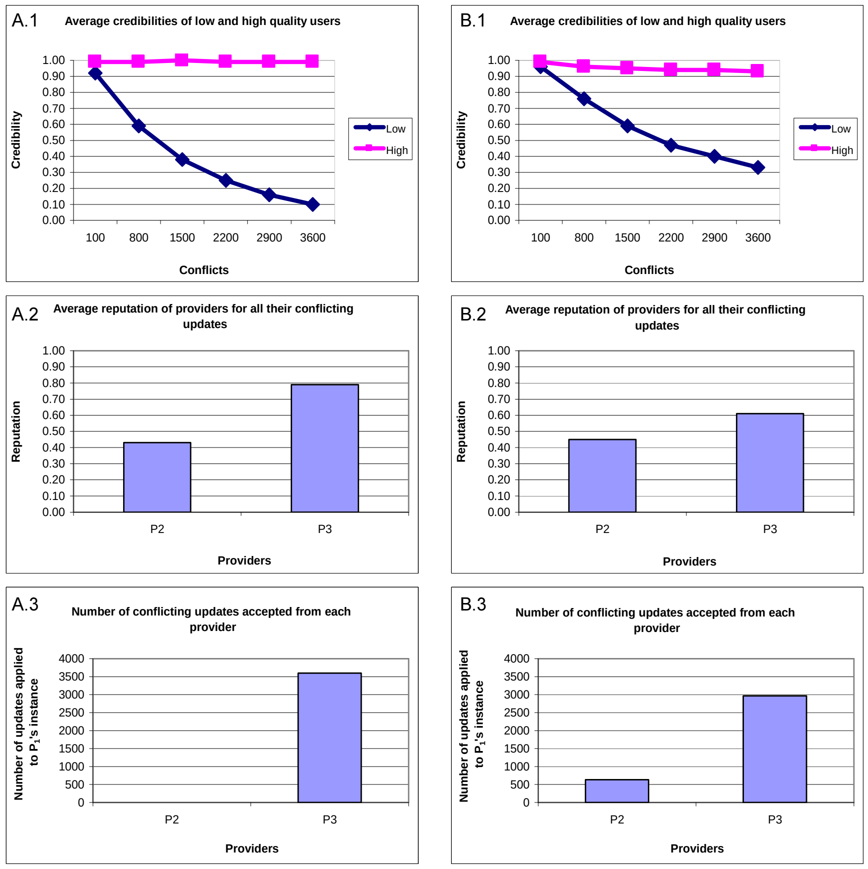


Figure 5.2: Experiment Results of Two Data Sets (A and B). A: Honest Raters Out-number Dishonest Raters. B: Dishonest Raters Equals to Honest Raters

by A), honest raters out-number dishonest ones. Fig. 5.2(A) shows the effect of this inequality in calculating raters' credibilities, providers' reputations, and thus the number of accepted updates per each provider. The average credibility of each group of users is shown in Fig. 5.2(A.1) with increasing number of conflicts, while Fig. 5.2(A.2) represents the average reputations of providers peers, and the number of accepted updates from each provider is shown in Fig. 5.2(A.3). Because there are more honest raters, we can see that the average assessed reputation for each provider is almost identical to their actual behaviors. Moreover, the average credibility of honest raters is always high compared to that of dishonest group where it is drastically decreasing for consecutive conflicts. The result of the second set where the number of honest and dishonest raters are equal is shown in Fig. 5.2(B). This equality results in the dishonest raters' ratings forming the majority rating on several occasions. Therefore, we see an increase in the updates of p_2 being accepted by p_1. This causes a degradation in the credibility of honest raters, since their opinion differs from the majority opinion, and an increment in the dishonest raters' credibilities (Fig. 5.2(B.1)).

5.4.2 Multiple Consumers and Providers

In the second set of experiments, we developed a CDSS of 40 participant peers, where 20 of them are only providers, and the other 20 peers are only consumers, with each consumer peer having 20 local users. We have divided the provider peers into 5 different behavioral groups that represent the real life scenarios: providers that always perform with consistently high quality (i.e., their updates are correct and of high value), providers that always perform with consistently low quality, providers that perform high at the beginning but start performing low after the time instance 200, providers that perform low at the beginning but they start performing high after the time instance 200, and the final group of providers that perform in a random manner, oscillating between high and low performance quality. We ran several experiments to cover the above mentioned CDSS cases, where each experiment is run multiple times for each scenario, and the averaged results over those runs are presented in the following.

The experiment rounds starts at time instance 0 and finish at time instance 400. The databases of all peers are empty at time instance 0. At the beginning of each time instance, all provider peers insert a new single update to their local instances, and then they publish their most recent update to others. The inserted

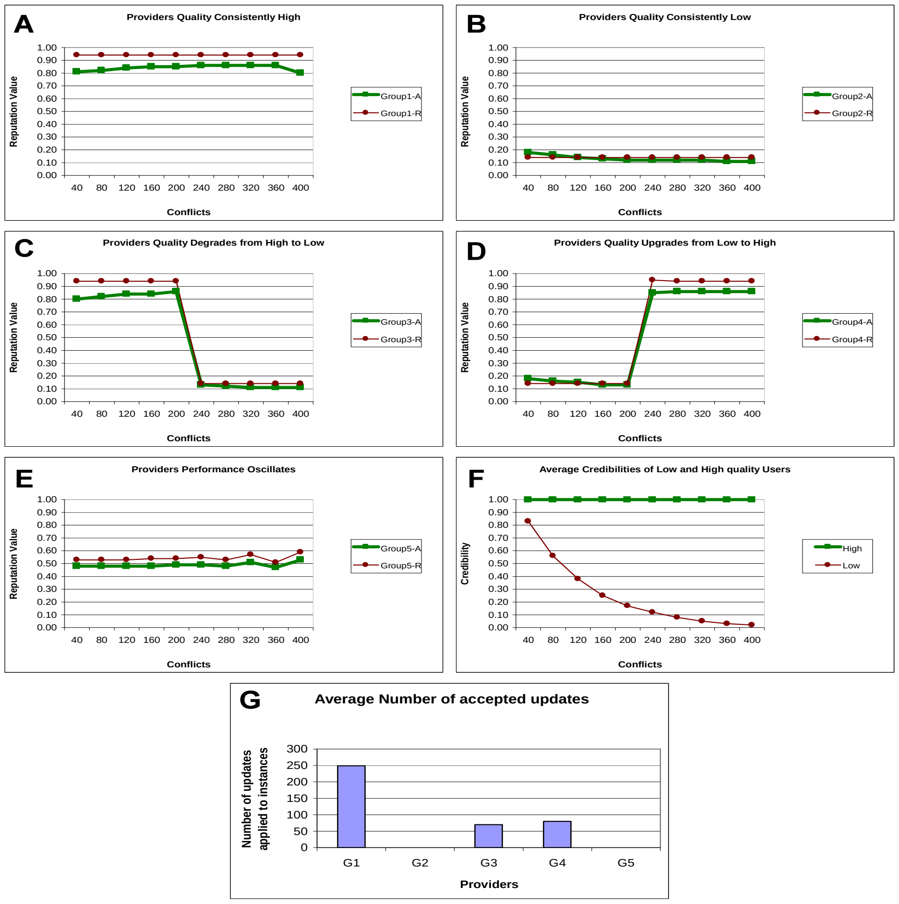


Figure 5.3: Reputation and Credibility Assessment: High Credibility users - 90%

updates at each time instance are almost identical. In other words, they all have the same value for the primary key attribute, but they have different values in at least one non-key attribute. In the same way, after all providers publish their most recent updates at a particular time instance, each consumer peer reconciles its local instance with the recently published updates. As all providers will publish conflicting updates at each time instance, a consumer peer will find that all imported updates conflict with each other at each reconciliation point. Thus, a new conflict group that contains all imported updates in this particular time instance is added to the deferred set of the reconciling peer. Provider peers are assigned degrees of quality or behavior in the following manner: it is in the range $[0.9, 1.0]$ for the first group, $[0.1, 0.2]$ for the second group, $[0.9, 1.0]$ for the third group in the first half of the experiment run and $[0.1, 0.2]$ in the second half, $[0.1, 0.2]$ for the fourth group in the first half of the experiment run and $[0.9, 1.0]$ in the second half, and $[0.1, 1.0]$ for the last group.

We further divided the experiments to model the different percentages of honest and dishonest users. In the interest of space, we present two cases in the following. In the first one, 90% of the users are high quality users (i.e., are honest), with values in the range $[0.8, 1.0]$, and 10% of them are low quality users in the range $[0.1, 0.2]$. In the other set, the percentage of high quality users is set to 60%. A high quality rater generates a rating that differs at most 10% from the actual value. In contrast, a low quality rater generates a value that differs at least by 75% from the actual rating value. At the beginning of the experiment rounds, we assume that all local users of the reconciling peers and all peers have their credibility values set to 1.0 (i.e., the maximum credibility value).

The plots (A-E) in Fig. 5.3 and Fig. 5.4 show the effect of the size of low quality raters in calculating the reputation values of each provider group. Each plot, from A to E, shows the comparison between the average of actual provider group quality (GroupX-R) and the average of assessed provider group reputation (GroupX-A). Similarly, plot F shows the comparison between the average credibility values of high and low quality user groups in all consumer peers. The last plot (G) shows the average number of updates accepted by all consumer peers from each group of providers.

It can be seen from Fig. 5.3 that when the percentage of low quality users is only 10% of the total number of local users, the computed assessed reputation values are almost equal to the original provider behavior. This is expected because Low quality users' behavior is captured and their credibilities are thus reduced (Fig. 5.3.F), which means that their provided ratings are also decreased. Fig. 5.3.G shows

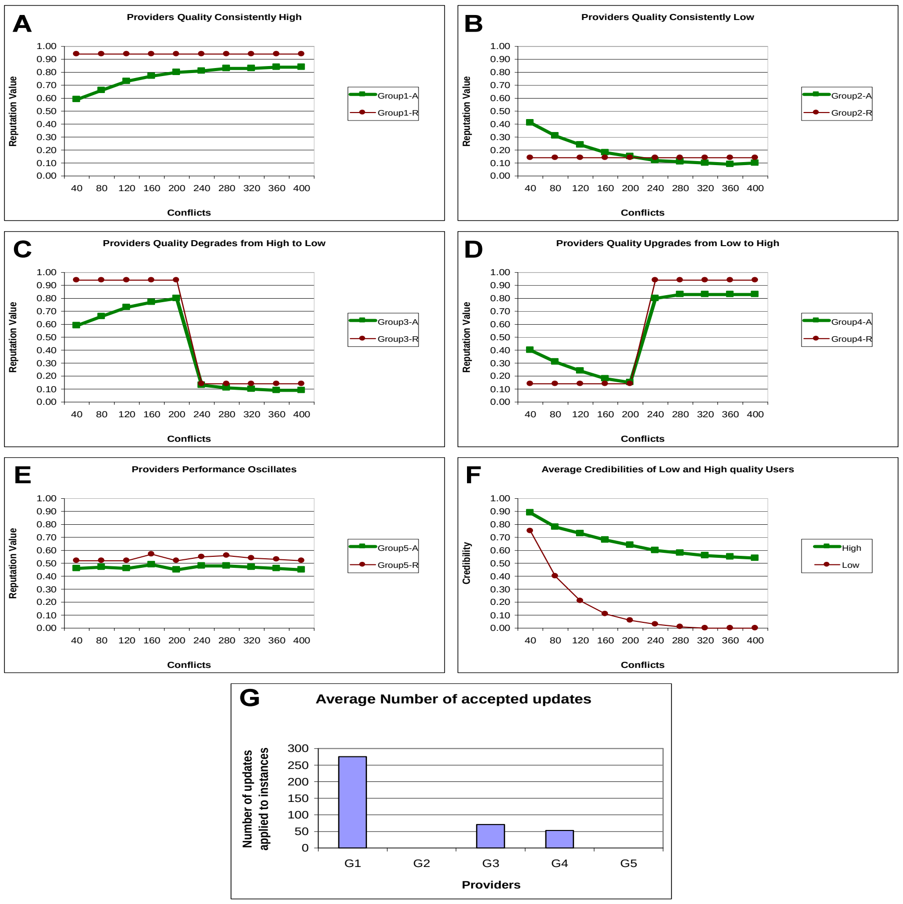


Figure 5.4: Reputation and Credibility Assessment: High Credibility users - 60%

the average number of updates accepted by reconciling peers from each group. The chance of accepting updates from groups G_1 and G_3 are the same at the first half of the simulation time, while it is the same for groups G_1 and G_4 at the second half. We can see that there are no updates accepted from either group G_2 or group G_5, as the reputation values for members of these groups are low most of the time.

Fig. 5.4 shows the result of the second set, where 40% of users are low quality. We see from Fig. 5.4.F that credibilities of both low and high quality users are decreased, and thus the difference between actual and assessed reputation is high. But within the same time, credibilities of low quality users are still decreased more, which reduces the difference between actual and assessed reputation. The simulation results show that our approach can effectively assess the reputation of providers even when the percentage of low quality users reaches 40% of the total number of users.

5.4.3 Execution Time Comparison

We also evaluate the execution time of our approach in comparison to the Orchestra system (Taylor and Ives, 2006) (a primary CDSS). In orchestra, reconciliation is not trust-based, and transactions are applied (i.e. reconciled) if they satisfy a given set of requirements. Others are either deferred or rejected. Fig. 5.5(a) shows the execution times for an average peer with one transaction. Here we assume that a distributed storage scheme is followed, where “requests to follow antecedent transaction chains dominate the running time” (Taylor and Ives, 2006). We can see that ACR’s running time is slightly higher than Orchestra, due to the number of trust-messages exchanged in addition to the normal updates. In either case, frequent reconciliations put a heavier load on overall system resources, potentially reducing performance. Similarly, Fig. 5.5(b) shows the execution times with increasing number of participant peers. We can see that with a higher number of peers, more transactions need to considered and compared. This automatically increases the number of trust-messages across the network, and thereby the total reconciliation time. However, we posit that the automated reconciliation that ACR provides, with better accuracy, justifies the slightly higher running times.

5.5 Related Work

In this section, we provide a brief overview of related literature on conflict resolution and trust management in peer-oriented environments. Approaches for the problem of inconsistent data have been described in detail in the context of traditional data

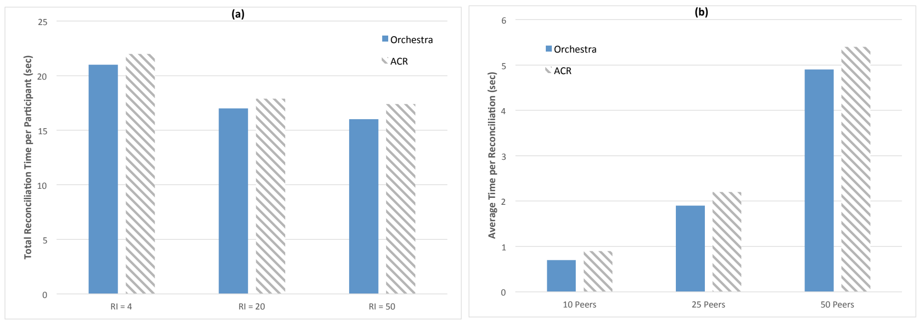


Figure 5.5: Execution Times with: (a) Variable Reconciliation Interval (with transaction size at one). RI is the number of transactions published between reconciliations. (b) Variable Number of Peers

integration systems. For instance, (Naumann et al., 2006), (Bleiholder and Naumann, 2006), (Bilke et al., 2005), (Motro and Anokhin, 2006), (Bleiholder et al., 2007), and (Motro and Anokhin, 2004) described different approaches to conflict resolution while integrating heterogeneous database sources (See (Bleiholder and Naumann, 2009) for a comprehensive survey about conflict classifications, strategies, and systems in heterogeneous sources).

Approaches for handling conflicts in community shared databases, based on the concept of multi-versioned database, are described in (Gatterbauer et al., 2009), (Pichler et al., 2010), and (Gatterbauer and Suciu, 2010). In (Gatterbauer et al., 2009), a BeliefDB system enables users to annotate existing data or even existing annotations, by adding their own beliefs that may agree or disagree with exiting data or annotations. A belief database contains both base data in the form of tuples and belief statements that annotate these tuples. It also represents a set of belief worlds, where each world belongs to a different user. Moreover, a belief-aware query language is introduced to represent queries over a belief database. This query language can be used to retrieve facts that are believed or not believed by a particular user. It also can be used to query for the agreements or disagreements on particular facts between users. An algorithm is also proposed in (Gatterbauer et al., 2009) to translate belief database queries into equivalent relational database queries.

Ref. (Gatterbauer and Suciu, 2010) describes an automatic conflict resolution based on trust mappings between users. A user usually has trust relationships with other users in the community. A user also assigns different trust priorities for different trusted users. To resolve a conflicting data, a user accepts a data value that comes from the most trusted user. Thus, each user is shown his own consistent version of the shared database based on his trust mappings and priorities with other users.

Ref. (Pichler et al., 2010) handles inconsistent data by allowing users to rate data. Updates done by users are stored in a shared, uncertain database, where all versions of conflicting updates are inserted into the database in parallel. In other words, all update operations, whether insertion, replacement, or deletion, are treated as insertion operations. Users in (Pichler et al., 2010) can update, query, and even rate the quality of updates, based on their own beliefs. The rating is usually weighted according to the reputation of the user who does the rating. Conflicting updates are usually various versions of the same tuple, sharing the same key, but having different values for non-key attributes. For each version of a tuple, the ratings of different users are collected, and the average rating for this version is computed. The reputation of a user who initiates the rated update can be then computed by comparing aggregate ratings of his updates to aggregate ratings of others. The computation of a user's reputation is incrementally, such that a new reputation value is computed for the user each time a new rating arrives. For answering a query from a user, the average rating of each consistent version (or world) of the database is computed, and the best rated world is found. After that, a user query is answered according to this consistent version of the database.

The work done in (Pichler et al., 2010) is similar to that of (Gatterbauer et al., 2009) (Gatterbauer and Suciu, 2010), in that all apply a multi-versioned database model to resolve conflicts. However, each user in (Gatterbauer et al., 2009) (Gatterbauer and Suciu, 2010), based on his own beliefs or trust mappings, sees his own consistent version of the shared database. In contrast, all users in (Pichler et al., 2010) see the most consistent version of the database which has the best rating. Our approach is similar to (Pichler et al., 2010) in that it also deploys community feedback to resolve conflicts. However, we only deploy the community feedback for the purpose of resolving conflicts between the updates of conflict groups in the deferred set of a local peer. Moreover, our approach is based on the CDSS, where each participant peer maintains a relational and consistent database instance, where conflicts between data are not allowed due to the restrictions of the relational DBMS. On the other hand, (Pichler et al., 2010) deploys the concept of uncertain and multi-versioned database, such that all conflicting updates are kept permanently in the same database, and users' queries are answered based on the combination of updates that have the best rating.

5.6 Conclusion and Future Work

We presented an approach to resolve conflicts that may arise due to the propagation of updates among related peers in a CDSS. The focus is to resolve conflicts in the deferred set (of a CDSS's reconciling peer) by collecting feedbacks about the quality of the conflicting updates from the local community (i.e., local users) and remote peers. When a new conflict group is added to the deferred set of a reconciling peer, it first inquires the participant remote peers about their experience while dealing with the provider peers that have updates in this particular conflict group. Then, for each provider peer in the conflict group, the reconciling peer aggregates the rating values received from remote raters to compute the remote assessed reputation value ($RRPP$) of the provider peer. Second, after a new conflict group is added to the deferred set of a reconciling peer, local users also rate the provider peers that have updates in this particular conflict group according to the quality of their updates. It then computes the local assessed reputation ($LRPP$) for each provider peer in the conflict group. Last, the assessed reputation of each provider peer in the conflict group is aggregated by weighting both $RRPP$ and $LRPP$ values. Thus, the reconciling peer can resolve the conflict in a conflict group by accepting and applying the update that comes from the provider peer with the highest reputation value to its local instance, provided it does not violate its state. All other updates in the conflict group are rejected. Experiment results suggest that the CDSS can be extended with very little overhead (in terms of execution time) to automatically and efficiently resolve conflicts that may arise during the reconciliation operation of a participant peer. We plan to extend this work, to utilize community feedbacks not only to resolve conflicts for the updates in the deferred set, but also to deploy community feedbacks for the purpose of automatically defining trust policies for the local peer, thereby omitting the role of the administrator in defining trust policies.

APPENDIX A: Helper functions for extracting *Ontop* mappings

ALGORITHM E.0.1: GenerateSqlQuery.

```
Function GenerateSqlQuery($T_i$, $ruletype$, $\mathcal{K}_F$)
    if $ruletype$ is null then
        $sqlSelect \leftarrow$ "$*$";
        $sqlFrom \leftarrow T_i.name$;
        $sqlWhere \leftarrow null$;
    else if $ruletype$ is "DEPENDENT" then
        $sqlQuery \leftarrow$ GenerateSqlQueryForDependentRule($T_i$, $\mathcal{K}_F$) ;
    else if $ruletype$ is "RECURSIVE" then
        $sqlQuery \leftarrow$ GenerateSqlQueryForRecursiveRule($T_i$, $\mathcal{K}_F$) ;
    else if $ruletype$ is "NARYJOIN" then
        $sqlQuery \leftarrow$ GenerateSqlQueryForNaryJoinRule($T_i$) ;
    if $sqlWhere$ is null then
        $sqlQuery \leftarrow$ "SELECT " + $sqlSelect$ + " FROM " + $sqlFrom$;
    else
        $sqlQuery \leftarrow$ "SELECT " + $sqlSelect$ + " FROM " + $sqlFrom$+
         "WHERE " + $sqlWhere$;
    return $sqlQuery$;
```

ALGORITHM E.0.2: GenerateSqlQueryForDependentRule.

```
Function GenerateSqlQueryForDependentRule($T_i$, $\mathcal{K}_F$)
    $T_j \leftarrow$ GetReferencedTable($\mathcal{K}_F$);
    $sqlSelect \leftarrow null$;
    $sqlFrom \leftarrow T_i.name$ + " , " + $T_j.name$;
    $sqlWhere \leftarrow null$;
    foreach $\mathcal{K}_p \in \mathcal{PK}_i$ do
        $sqlSelect \leftarrow$
        $sqlSelect + T_i.name +$ "." $+ \mathcal{K}_p.name +$ " AS " $+ T_i.name +$ "_" $+ \mathcal{K}_p.name +$ ", "
    foreach $a_i \in \mathcal{K}_F$ do
        $sqlWhere \leftarrow sqlWhere + T_i.name +$ "." $+ a_i.name +$ " = ";
        $a_j \leftarrow$ GetReferencedColumn($a_i$);
        $sqlSelect \leftarrow$
        $sqlSelect + T_j.name +$ "." $+ a_j.name +$ " AS " $+ T_j.name +$ "_" $+ a_j.name$;
        $sqlWhere \leftarrow sqlWhere + T_j.name +$ "." $+ a_j.name$;
        if $a_i$ is not last attribute in $\mathcal{K}_F$ then
            $sqlSelect \leftarrow sqlSelect +$ ", ";
            $sqlWhere \leftarrow sqlWhere +$ " AND ";
    $sqlQuery \leftarrow$ "SELECT " $+ sqlSelect +$ " FROM " $+ sqlFrom +$
     "WHERE " $+ sqlWhere$;
    return $sqlQuery$;
```

ALGORITHM E.0.3: GenerateSqlQueryForRecursiveRule.

```
Function GenerateSqlQueryForRecursiveRule($T_i$, $\mathcal{K}_F$)
    $sqlSelect \leftarrow null$;
    $sqlWhere \leftarrow null$;
    $sqlFrom \leftarrow T_i.name$ + " " + $T_i.name$ + "_child" + " , " + ;
     $Ti.name$ + " " + $T_i.name$ + "_parent";
    foreach $\mathcal{K}_p \in \mathcal{PK}_i$ do
        $sqlSelect \leftarrow$
        $sqlSelect + T_i.name +$ "_child." $+ \mathcal{K}_p.name +$ " AS " $+ T_i.name +$ "_child_" $+ \mathcal{K}_p.name +$ ", ";
    foreach $a_i \in \mathcal{K}_F$ do
        $sqlWhere \leftarrow sqlWhere$ + $T_i.name$ + "_child." + $a_i.name$ + " = ";
        $a_j \leftarrow$ GetReferencedColumn($a_i$);
        $sqlSelect \leftarrow$
        $sqlSelect + T_i.name +$ "_parent." $+ \mathcal{K}_p.name +$ " AS " $+ T_i.name +$ "_parent_" $+ \mathcal{K}_p.name$;

        $sqlWhere \leftarrow sqlWhere$ + $T_j.name$ + "_parent." + $a_j.name$;
        if $a_i$ is not last attribute in $\mathcal{K}_F$ then
            $sqlSelect \leftarrow sqlSelect$ + ", ";
            $sqlWhere \leftarrow sqlWhere$ + " AND ";
    $sqlQuery \leftarrow$ "SELECT " $+ sqlSelect +$ " FROM " $+ sqlFrom +$
     "WHERE " $+ sqlWhere$;
    return $sqlQuery$;
```

ALGORITHM E.0.4: GenerateSqlQueryForNaryJoinRule.

```
1  Function GenerateSqlQueryForNaryJoinRule($T_i$)
2      $NaryJoinFKs \leftarrow \phi$ ;
3      foreach $\mathcal{K}_F \in \mathcal{FK}_i$ do
4          if $Attributes(\mathcal{K}_F) \in \mathcal{PK}_i$ then
5              $NaryJoinFKs \leftarrow NaryJoinFKs \cup \mathcal{K}_F$;
6      $sqlSelect \leftarrow null$;
7      $sqlFrom \leftarrow T_i.name$;
8      $sqlWhere \leftarrow null$;
9      foreach $\mathcal{K}_p \in \mathcal{PK}_i$ do
10         $sqlSelect \leftarrow$
           $sqlSelect + T_i.name + “.” + \mathcal{K}_p.name + “\ AS\ ” + T_i.name + “\_” + \mathcal{K}_p.name + “,\ ”$;
11     foreach $\mathcal{K}_F \in NaryJoinFKs$ do
12         $T_j \leftarrow$ GetReferencedTable($\mathcal{K}_F$);
13         $sqlFrom \leftarrow sqlFrom\ +\ “\ ,\ ”\ +\ T_j.name$;
14         foreach $a_i \in \mathcal{K}_F$ do
15             $sqlWhere \leftarrow sqlWhere\ +\ T_i.name\ +\ “.”\ +\ a_i.name\ +\ “\ =\ ”$;
16             $a_j \leftarrow$ GetReferencedColumn($a_i$);
17             $sqlSelect \leftarrow$
               $sqlSelect + T_j.name + “.” + a_j.name + “\ AS\ ” + T_j.name + “\_” + a_j.name$;
18             $sqlWhere \leftarrow sqlWhere\ +\ T_j.name\ +\ “.”\ +\ a_j.name$;
19             if $a_i$ is not last attribute in $\mathcal{K}_F$ OR $\mathcal{K}_F$ is not last key in $NaryJoinFKs$
               then
20                 $sqlSelect \leftarrow sqlSelect\ +\ “,\ ”$;
21                 $sqlWhere \leftarrow sqlWhere\ +\ “\ AND\ ”$;
22     $sqlQuery \leftarrow “SELECT\ ” + sqlSelect + “\ FROM\ ” + sqlFrom +$
23      $“WHERE\ ” + sqlWhere$;
24     return $sqlQuery$;
```

ALGORITHM E.0.5: GenerateSubject.

```
Function GenerateSubject($T_i$, $T_j$, $ruleType$, $subjectType$)
    $subject \leftarrow$ “ : ” + $T_i$ + “/”;
    foreach $a_i \in \mathcal{PK}_i$ do
        $subject \leftarrow subject$ + $a_i.name$ + “ = ”;
        if $ruleType$ is null then
            $subject \leftarrow subject$ + “{” + $a_i.name$ + “}”;
        else if $ruleType$ is “DEPENDENT” then
            $subject \leftarrow subject$ + “{” + $T_i$ + “_” + $a_i.name$ + “}”;
        else if $ruleType$ is “RECURSIVE” and $subjectType$ is “DOMAIN” then
            $subject \leftarrow subject$ + “{” + $T_i$ + “_child_” + $a_i.name$ + “}”;
        else if $ruleType$ is “RECURSIVE” and $subjectType$ is “RANGE” then
            $subject \leftarrow subject$ + “{” + $T_i$ + “_parent_” + $a_i.name$ + “}”;
        else if $ruleType$ is “BINARYJOIN” then
            // Get attribute $a_j$ in $T_j$ that is referring to $a_i$ in $T_i$.
            $a_j \leftarrow$ GetReferringColumn($a_i$, $T_i$, $T_j$);
            $subject \leftarrow subject$ + “{” + $a_j.name$ + “}”;
        if $a_i$ is not last attribute in $\mathcal{PK}_i$ then
            $subject \leftarrow subject$ + “; ”;
    return $subject$;
```

ALGORITHM E.0.6: GenerateClassTriple.

```
Function GenerateSubject($T_i$, $sub$)
    $triple \leftarrow sub$ + “ a ” + “ : ” + $T_i$ + “ ; ”;
    return $triple$;
```

ALGORITHM E.0.7: GenerateDataPropertyObject.

```
Function GenerateDataPropertyObject($T_i$, $a_j$)
    $numeric \leftarrow$
    $\{INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT,$
    $DECIMAL, NUMERIC, FLOAT, DOUBLE\}$;
    if $a_i.type \in numeric$ then
        $object \leftarrow$ “{” + $a_j.name$ + “}” + “^^” + SqlToXsdDataType($a_j$);
    else
        $object \leftarrow$ “{” + $a_j.name$ + “}”;
    return $object$;
```

ALGORITHM E.0.8: GenerateDataPropertyPredicate.

```
Function GeneratePropertyPredicate($T_i$, $a_j$)
    $predicate \leftarrow$ “ : ” + $T_i$ + “#” + $a_j$;
    return $predicate$;
```

ALGORITHM E.0.9: GenerateDataPropertyTriple.

```
Function GenerateDataPropertyTriple($T_i$, $sub$, $predicate$, $obj$)
    $triple \leftarrow sub$ + “ ” + $predicate$ + “ ” + $obj$;
    return $triple$;
```

ALGORITHM E.0.10: CreateObjectPropertyName.

```
Function CreateObjectPropertyName($table$, $prefix$, $postfix$)
    $name \leftarrow table$ + “#” + $prefix$ + $postfix$;
    return $name$;
```

ALGORITHM E.0.11: GenerateObjectPropertyPredicate.

```
Function GenerateObjectPropertyPredicate($T_i$, $OP_i$)
    $predicate \leftarrow$ “ : ” + $T_i$ + “#” + $OP_i$;
    return $predicate$;
```

ALGORITHM E.0.12: GenerateObjectPropertyTriple.

```
Function GenerateObjectPropertyTriple($T_i$, $sub$, $predicate$, $obj$)
    $triple \leftarrow sub$ + “ ” + $predicate$ + “ ” + $obj$ + “ .”;
    return $triple$;
```

ALGORITHM E.0.13: GenerateObjectPropertyTripleForNaryJoin.

```
Function GenerateObjectPropertyTripleForNaryJoin($\mathcal{K}_F$, $sub_i$, $predicate_i$, $triple$)
    $T_j \leftarrow$ GetReferencedTable($\mathcal{K}_F$);
    $object_j \leftarrow$ GenerateSubject($T_j$, $null$, $null$, $null$);
    if $triple$ is $null$ then
        $triple \leftarrow sub_i$ + “ ” + $predicate_i$ + “ ” + $object_j$;
    else
        $triple \leftarrow triple$ + “ , ” + $object_j$;
    return $triple$;
```

ALGORITHM E.0.14: SqlToXsdDataType.

```
1  Function SqlToXsdDataType($a_i$)
2      if $a_i.type \in \{CHAR, VARCHAR, BINARY, BLOB, TEXT\}$ then
3          $xsdType \leftarrow$ "$rdfs : literal$";
4      else if $a_i.type \in$
       $\{INTEGER, INT, SMALLINT, TINYINT, MEDIUMINT, BIGINT\}$ then
5          $xsdType \leftarrow$ "$xsd : integer$";
6      else if $a_i.type \in \{DECIMAL, NUMERIC\}$ then
7          $xsdType \leftarrow$ "$xsd : decimal$";
8      else if $a_i.type \in \{FLOAT, DOUBLE\}$ then
9          $xsdType \leftarrow$ "$xsd : double$";
10     else if $a_i.type \in \{DATETIME, TIMESTAMP\}$ then
11         $xsdType \leftarrow$ "$xsd : datetime$";
12     return $xsdType$;
```

APPENDIX B: List of Publications

Journal Publications

1. F. Khazalah, Z. Malik, and A. Rezgui, “Automated Conflict Resolution in CDSS using Community Feedbacks”, *Information Sciences*, vol. 298, pp. 407–424, March 20, 2015.

Conference Publications

1. F. Khazalah, Z. Malik, and A. Rezgui, “Automatic Mapping Rules and OWL Ontology Extraction for the Ontop”, 10th IEEE International Conference on Collaborative Computing: Networking, Applications and Worksharing (CollaborateCom), Oct. 22-25, 2014, Miami, FL, USA.

2. F. Khazalah, B. Medjahed, and Z. Malik, “Automatic Conflict Resolution in a CDSS”, 24th International Conf. on Scientific and Statistical Database Management (SSDBM 2012), 25-27 June 2012, Greece.

3. F. Khazalah, I. Ababneh, and Z. Malik, “Forwarding Group Maintenance of ODMRP in MANETs”, 4th International Conference on Ambient Systems, Networks and Technologies, 25-28 June 2013, Halifax, Nova Scotia, Canada.

4. E. Najmi, K. Hashmi, F. Khazalah and Z. Malik, “Intelligent Semantic Question Answering System”, IEEE International Conf on Cybernetics, Lausanne Switzerland, 13-15 June 2013.

5. M. Abi-Antoun, N. Ammar, and F. Khazalah, “A case Study in Adding Ownership Domain Annotations”, Wayne State University Technical Report 2010.

REFERENCES

A. Doan, J. Madhavan, P. Domingos, and A. Halevy, “Learning to map between ontologies on the semantic web,” in *Proceedings of the 11th international conference on World Wide Web*. ACM, 2002, pp. 662–673.

D. Beckett and J. Grant, “SWAD-Europe: Mapping Semantic Web Data with RDBMSes,” W3C, W3C Report, jan 2003, http://www.w3.org/2001/sw/Europe/reports/scalable_rdbms_mapping_report/.

M. del Mar Roldan-Garcia and J. F. Aldana-Montes, “A survey on disk oriented querying and reasoning on the semantic web,” in *Proceedings of the 22nd International Conference on Data Engineering Workshops*. IEEE Computer Society, 2006, p. p. 58.

K. C.-C. Chang, B. He, C. Li, M. Patel, and Z. Zhang, “Structured databases on the web: Observations and implications,” *ACM SIGMOD Record*, vol. 33, no. 3, pp. 61–70, 2004.

J. Geller, S. A. Chun, and Y. J. An, “Toward the semantic deep web.” *IEEE Computer*, vol. 41, no. 9, pp. 95–97, 2008.

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, 2012.

J. F. Sequeda, R. Depena, and D. P. Miranker, “Ultrawrap: Using sql views for rdb2rdf,” *Proc. of ISWC2009*, 2009.

A. Poggi, D. Lembo, D. Calvanese, G. De Giacomo, M. Lenzerini, and R. Rosati, “Linking data to ontologies,” in *Journal on data semantics X*. Springer, 2008, pp. 133–173.

M. Rodriguez-Muro and D. Calvanese, “Quest, an owl 2 ql reasoner for ontology-based data access,” in *Proc. of the 9th Int. Workshop on OWL: Experiences and Directions (OWLED 2012)*, ser. CEUR Electronic Workshop Proceedings, http://ceur-ws.org/, vol. 849, 2012.

M. Rodrıguez-Muro, R. Kontchakov, and M. Zakharyaschev, “Ontop at work,” in *Proc. of the 10th OWL: Experiences and Directions Workshop (OWLED 2013).*, 2013.

M. Rodriguez-Muro, M. Rezk, J. Hardi, M. Slusnys, T. Bagosi, and D. Calvanese, “Evaluating sparql-to-sql translation in ontop,” 2013.

R. Cyganiak, S. Sundara, and S. Das, “R2RML: RDB to RDF mapping language,” W3C, W3C Recommendation, sep 2012, http://www.w3.org/TR/2012/REC-r2rml-20120927/.

M. Rodriguez-Muro, L. Lubyte, and D. Calvanese, “Realizing ontology based data access: A plug-in for protégé,” in *Data Engineering Workshop, 2008. ICDEW 2008. IEEE 24th International Conference on.* IEEE, 2008, pp. 286–289.

B. Motik, B. Parsia, and P. Patel-Schneider, “OWL 2 web ontology language structural specification and functional-style syntax (second edition),” W3C, W3C Recommendation, dec 2012, http://www.w3.org/TR/2012/REC-owl2-syntax-20121211/.

G. Antoniou, *A semantic Web primer.* Cambridge, Mass: MIT Press, 2008.

D. Wood, M. Lanthaler, and R. Cyganiak, “RDF 1.1 concepts and abstract syntax,” W3C, W3C Recommendation, feb 2014, http://www.w3.org/TR/2014/REC-rdf11-concepts-20140225/.

R. Guha and D. Brickley, "RDF schema 1.1," W3C, W3C Recommendation, feb 2014, http://www.w3.org/TR/2014/REC-rdf-schema-20140225/.

E. Prud'hommeaux and A. Seaborne, "SPARQL query language for RDF," W3C, W3C Recommendation, jan 2008, http://www.w3.org/TR/2008/REC-rdf-sparql-query-20080115/.

N. Konstantinou, D.-E. Spanos, P. Stavrou, and N. Mitrou, "Technically approaching the semantic web bottleneck," *International Journal of Web Engineering and Technology*, vol. 6, no. 1, pp. 83–111, 2010.

M. Rodriguez-Muro, J. Hardi, and D. Calvanese, "Quest: Efficient sparql-to-sql for rdf and owl," in *Demos of the 12th Int. Semantic Web Conf. (ISWC 2012)*, 2012.

D. Calvanese, G. De Giacomo, D. Lembo, M. Lenzerini, A. Poggi, and R. Rosati, "Ontology-based database access," in *Proceedings of the Fifteenth Italian Symposium on Advanced Database Systems, SEBD, 17-20 June 2007, Torre Canne, Fasano, BR, Italy*, 2007, pp. 324–331.

C. Bizer and A. Seaborne, "D2rq-treating non-rdf databases as virtual rdf graphs," in *Proceedings of the 3rd international semantic web conference (ISWC2004)*, 2004.

O. Erling and I. Mikhailov, "Virtuoso: Rdf support in a native rdbms," in *Semantic Web Information Management*. Springer, 2010, pp. 501–519.

S. Auer, S. Dietzold, J. Lehmann, S. Hellmann, and D. Aumueller, "Triplify: lightweight linked data publication from relational databases," in *Proceedings of the 18th international conference on World wide web*. ACM, 2009, pp. 621–630.

M. Fisher, J. Ellis, and J. C. Bruce, *JDBC API Tutorial and Reference*, 3rd ed. Pearson Education, 2003.

I. Astrova, “Rules for mapping sql relational databases to owl ontologies,” in *Metadata and Semantics*. Springer, 2009, pp. 415–424.

A. Buccella, M. R. Penabad, F. J. Rodriguez, A. Farina, and A. Cechich, “From relational databases to owl ontologies,” in *Proceedings of the 6th National Russian Research Conference*, 2004.

K. M. Albarrak and E. H. Sibley, “Translating relational & object-relational database models into owl models,” in *Information Reuse & Integration, 2009. IRI’09. IEEE International Conference on*. IEEE, 2009, pp. 336–341.

L. Lubyte and S. Tessaris, “Automatic extraction of ontologies wrapping relational data sources,” in *Database and Expert Systems Applications*. Springer, 2009, pp. 128–142.

K. Sonia and S. Khan, “R2o transformation system: relation to ontology transformation for scalable data integration,” in *Proceedings of the 2008 international symposium on Database engineering & applications*. ACM, 2008, pp. 291–295.

F. Cerbah, “Mining the content of relational databases to learn ontologies with deeper taxonomies,” in *Web Intelligence and Intelligent Agent Technology, 2008. WI-IAT’08. IEEE/WIC/ACM International Conference on*, vol. 1. IEEE, 2008, pp. 553–557.

C. Curino, G. Orsi, E. Panigati, and L. Tanca, “Accessing and documenting relational databases through owl ontologies,” in *Flexible Query Answering Systems*. Springer, 2009, pp. 431–442.

N. Alalwan, H. Zedan, and F. Siewe, "Generating owl ontology for database integration," in *Advances in Semantic Processing, 2009. SEMAPRO'09. Third International Conference on.* IEEE, 2009, pp. 22–31.

S. H. Tirmizi, J. Sequeda, and D. Miranker, "Translating sql applications to the semantic web," in *Database and Expert Systems Applications.* Springer, 2008, pp. 450–464.

C. P. de Laborda and S. Conrad, "Relational. owl: a data and schema representation format based on owl," in *Proceedings of the 2nd Asia-Pacific conference on Conceptual modelling-Volume 43.* Australian Computer Society, Inc., 2005, pp. 89–96.

N. Konstantinou, D.-E. Spanos, and N. Mitrou, "Ontology and database mapping: a survey of current implementations and future directions," *Journal of Web Engineering*, vol. 7, no. 1, pp. 1–24, 2008.

R. Ghawi and N. Cullot, "Database-to-ontology mapping generation for semantic interoperability," in *VDBL07 conference, VLDB Endowment ACM*, 2007, pp. 1–8.

J. Barrasa, Ó. Corcho, and A. Gómez-Pérez, "R2o, an extensible and semantically based database-to-ontology mapping language," in *Proceedings of the 2nd Workshop on Semantic Web and Databases (SWDB2004).* Springer-Verlag, 2004.

M. Hert, G. Reif, and H. C. Gall, "Updating relational data via sparql/update," in *Proceedings of the 2010 EDBT/ICDT Workshops*, ser. EDBT '10. New York, NY, USA: ACM, 2010, pp. 24:1–24:8.

A. Seaborne, G. Manjunath, C. Bizer, J. Breslin, S. Das, I. Davis, S. Harris, K. Idehen, O. Corby, K. Kjernsmo *et al.*, "Sparql/update: A language for updating rdf graphs," *W3C Member Submission*, vol. 15, 2008, http://www.w3.org/Submission/2008/SUBM-SPARQL-Update-20080715/.

A. Passant, P. Gearon, and A. Polleres, "SPARQL 1.1 update," W3C, W3C Recommendation, mar 2013, http://www.w3.org/TR/2013/REC-sparql11-update-20130321/.

V. Eisenberg and Y. Kanza, "D2rq/update: updating relational data via virtual rdf," in *Proceedings of the 21st international conference companion on World Wide Web*. ACM, 2012, pp. 497–498.

W. Gatterbauer, M. Balazinska, N. Khoussainova, and D. Suciu, "Believe it or not: adding belief annotations to databases," *Proc. VLDB Endow.*, vol. 2, pp. 1–12, August 2009.

R. Overbeek, T. Disz, and R. Stevens, "The seed: a peer-to-peer environment for genome annotation," *Commun. ACM*, vol. 47, pp. 46–51, November 2004.

P. Buneman, A. Chapman, and J. Cheney, "Provenance management in curated databases," in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, ser. Proc. SIGMOD. NY, USA: ACM, 2006, pp. 539–550.

A. Bairoch, R. Apweiler, C. H. Wu, W. C. Barker, B. Boeckmann, S. Ferro, E. Gasteiger, H. Huang, R. Lopez, M. Magrane, M. J. Martin, D. A. Natale, C. O'Donovan, N. Redaschi, and L.-S. L. Yeh, "The universal protein resource (uniprot)," *Nucleic Acids Research*, vol. 33, no. suppl 1, pp. D154–D159, 2005.

M. Tudor and K. Dvornich, "The naturemapping program: Resource agency environmental education reform," *Journal of Environmental Management*, vol. 32, no. 2, pp. 8–14, 2001.

C. Gouveia, A. Fonseca, A. Cmara, and F. Ferreira, “Promoting the use of environmental data collected by concerned citizens through information and communication technologies,” *Journal of Environmental Management*, vol. 71, no. 2, pp. 135–154, 2004.

W. Gatterbauer and D. Suciu, “Data conflict resolution using trust mappings,” in *Proc. SIGMOD*. NY, USA: ACM, 2010, pp. 219–230.

Z. G. Ives, N. Khandelwal, A. Kapur, and M. Cakir, “Orchestra: Rapid, collaborative sharing of dynamic data,” in *In CIDR*, 2005.

N. E. Taylor and Z. G. Ives, “Reconciling while tolerating disagreement in collaborative data sharing,” in *Proc. ACM SIGMOD*. NY, USA: ACM, 2006, pp. 13–24.

T. J. Green, G. Karvounarakis, Z. G. Ives, and V. Tannen, “Update exchange with mappings and provenance,” in *Proc. VLDB Endow.* VLDB Endowment, 2007, pp. 675–686.

Z. G. Ives, T. J. Green, G. Karvounarakis, N. E. Taylor, V. Tannen, P. P. Talukdar, M. Jacob, and F. Pereira, “The orchestra collaborative data sharing system,” *SIGMOD Rec.*, vol. 37, pp. 26–32, September 2008.

L. Kot and C. Koch, “Cooperative update exchange in the youtopia system,” *Proc. VLDB Endow.*, vol. 2, pp. 193–204, August 2009.

F. Naumann, A. Bilke, J. Bleiholder, and M. Weis, “Data fusion in three steps: resolving schema, tuple, and value inconsistencies,” *IEEE Data Engineering Bulletin*, pp. 21–31, 2006.

J. Bleiholder and F. Naumann, “Conflict handling strategies in an integrated information system,” in *Proc. IJCAI*, 2006.

A. Bilke, J. Bleiholder, F. Naumann, C. Böhm, K. Draba, and M. Weis, “Automatic data fusion with hummer,” in *Proc. of 31st Int. Conf. on VLDB*, ser. VLDB ’05. VLDB Endowment, 2005, pp. 1251–1254.

A. Motro and P. Anokhin, “Fusionplex: resolution of data inconsistencies in the integration of heterogeneous information sources,” *Information Fusion*, vol. 7, no. 2, pp. 176–196, 2006.

J. Bleiholder, K. Draba, and F. Naumann, “Fusem: exploring different semantics of data fusion,” in *Proc. VLDB*, ser. VLDB ’07. VLDB Endowment, 2007, pp. 1350–1353.

A. Motro and P. Anokhin, “Utility-based resolution of data inconsistencies,” in *Proc. IQIS*. NY, USA: ACM, 2004, pp. 35–43.

P. A. Bernstein, F. Giunchiglia, A. Kementsietsidis, J. Mylopoulos, L. Serafini, and I. Zaihrayeu, “Data management for peer-to-peer computing: A vision,” in *Proc. 5th International Workshop on the Web and Databases*, ser. WebDB ’02, 2002, pp. 89–94.

A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov, “Schema mediation in peer data management systems,” in *Data Engineering, 2003. Proceedings. 19th International Conference on*, march 2003, pp. 505–516.

R. Hull, “Managing semantic heterogeneity in databases: a theoretical prospective,” in *Proceedings of the sixteenth ACM SIGACT-SIGMOD-SIGART symposium on Principles of database systems*, ser. PODS ’97. New York, NY, USA: ACM, 1997, pp. 51–61.

J. D. Ullman, “Information integration using logical views,” in *Database Theory ICDT ’97*, ser. Lecture Notes in Computer Science, F. Afrati and P. Kolaitis, Eds. Springer Berlin, Heidelberg, 1997, vol. 1186, pp. 19–40.

A. Y. Halevy, “Answering queries using views: A survey,” *The VLDB Journal*, vol. 10, pp. 270–294, 2001.

L. Popa, Y. Velegrakis, M. A. Hernández, R. J. Miller, and R. Fagin, “Translating web data,” in *Proceedings of the 28th international conference on Very Large Data Bases*, ser. VLDB ’02. VLDB Endowment, 2002, pp. 598–609.

S. Melnik, P. A. Bernstein, A. Halevy, and E. Rahm, “Supporting executable mappings in model management,” in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, ser. SIGMOD ’05. New York, NY, USA: ACM, 2005, pp. 167–178.

M. Lenzerini, “Data integration: a theoretical perspective,” in *Proceedings of the twenty-first ACM SIGMOD-SIGACT-SIGART symposium on Principles of database systems*, ser. PODS ’02. New York, NY, USA: ACM, 2002, pp. 233–246.

E. Rahm and P. A. Bernstein, “On matching schemas automatically,” *VLDB Journal*, vol. 10, no. 4, pp. 334–350, 2001.

E. Rahm and P. A. Bernstein, “A survey of approaches to automatic schema matching,” *The VLDB Journal*, vol. 10, pp. 334–350, 2001.

A. Elmagarmid, P. Ipeirotis, and V. Verykios, “Duplicate record detection: A survey,” *Knowledge and Data Engineering, IEEE Transactions on*, vol. 19, no. 1, pp. 1–16, jan 2007.

V. I. Levenshtein, "Binary codes capable of correcting deletions, insertions, and reversals," *Soviet Physics Doklady*, vol. 10, no. 8, pp. 707–710, 1966.

A. Monge and C. Elkan, "The field matching problem: Algorithms and applications," in *In Proceedings of the Second International Conference on Knowledge Discovery and Data Mining*, 1996, pp. 267–270.

M. Cochinwala, V. Kurien, G. Lalk, and D. Shasha, "Efficient data reconciliation," *Information Sciences*, vol. 137, no. 1-4, pp. 1–15, 2001.

S. Tejada, C. A. Knoblock, and S. Minton, "Learning domain-independent string transformation weights for high accuracy object identification," in *Proceedings of the eighth ACM SIGKDD international conference on Knowledge discovery and data mining*, ser. KDD '02. New York, NY, USA: ACM, 2002, pp. 350–359.

S. Tejada, C. A. Knoblock, and S. Minton, "Learning object identification rules for information integration," *Information Systems*, vol. 26, no. 8, pp. 607–633, 2001.

J. R. Wang and S. E. Madnick, "The inter-database instance identification problem in integrating autonomous systems," in *Data Engineering, 1989. Proceedings. Fifth International Conference on*, 1989, pp. 46–55.

E.-P. Lim, J. Srivastava, S. Prabhakar, and J. Richardson.

A. Monge and C. Elkan, "An Efficient Domain-Independent Algorithm for Detecting Approximately Duplicate Database Records," in *SIGMOD workshop on data mining and knowledge discovery*, 1997.

D. Dey, S. Sarkar, and P. De, "Entity matching in heterogeneous databases: a distance-based decision model," in *System Sciences, 1998., Proceedings of the Thirty-First Hawaii International Conference on*, vol. 7, jan 1998, pp. 305–313.

J. Bleiholder and F. Naumann, “Data fusion,” *ACM Comput. Surv.*, vol. 41, pp. 1–41, January 2009.

D. Burdick, P. M. Deshpande, T. S. Jayram, R. Ramakrishnan, and S. Vaithyanathan, “Olap over uncertain and imprecise data,” in *Proceedings of the 31st international conference on Very large data bases*, ser. VLDB ’05. VLDB Endowment, 2005, pp. 970–981.

S. Chawathe, H. Garcia-Molina, J. Hammer, K. Ireland, Y. Papakonstantinou, J. Ullman, and J. Widom, “The tsimmis project: Integration of heterogeneous information sources,” in *IPSJ Conference*, 1994.

Y. Papakonstantinou, S. Abiteboul, and H. Garcia-molina, “Object fusion in mediator systems,” 1996.

H. Garcia-Molina, Y. Papakonstantinou, D. Quass, A. Rajaraman, Y. Sagiv, J. Ullman, V. Vassalos, and J. Widom, “The tsimmis approach to mediation: Data models and languages,” *Journal of Intelligent Information Systems*, vol. 8, pp. 117–132, 1997.

V. S. Subrahmanian, S. Adali, A. Brink, R. Emery, J. Lu, A. Rajput, T. Rogers, R. Ross, and C. Ward, “Intelligent caching in heterogeneous reasoning and mediator systems,” Tech. Rep., 1995.

R. Pichler, V. Savenkov, S. Skritek, and T. Hong-Linh, “Uncertain databases in collaborative data management,” in *Proc. VLDB Endow.* VLDB Endowment, 2010.

D. Maier and L. M. L. Delcambre, “Superimposed information for the internet,” in *WebDB (Informal Proceedings)*, 1999, pp. 1–9.

D. Srivastava and Y. Velegrakis, “Intensional associations between data and metadata,” in *Proceedings of the 2007 ACM SIGMOD international conference on Management of data*, ser. SIGMOD ’07. New York, NY, USA: ACM, 2007, pp. 401–412.

D. Bhagwat, L. Chiticariu, W.-C. Tan, and G. Vijayvargiya, “An annotation management system for relational databases,” *The VLDB Journal*, vol. 14, pp. 373–396, 2005.

P. Buneman, A. Chapman, and J. Cheney, “Provenance management in curated databases,” in *Proceedings of the 2006 ACM SIGMOD international conference on Management of data*, ser. SIGMOD ’06. New York, NY, USA: ACM, 2006, pp. 539–550.

P. Buneman, S. Khanna, and T. Wang-Chiew, “Why and where: A characterization of data provenance,” in *Database Theory ICDT 2001*, ser. Lecture Notes in Computer Science, J. Van den Bussche and V. Vianu, Eds. Springer Berlin, Heidelberg, 2001, vol. 1973, pp. 316–330.

L. Chiticariu, W.-C. Tan, and G. Vijayvargiya, “Dbnotes: a post-it system for relational databases based on provenance,” in *Proceedings of the 2005 ACM SIGMOD international conference on Management of data*, ser. SIGMOD ’05. New York, NY, USA: ACM, 2005, pp. 942–944.

F. Geerts and J. Van Den Bussche, “Relational completeness of query languages for annotated databases,” in *Proceedings of the 11th international conference on Database programming languages*, ser. DBPL’07. Berlin, Heidelberg: Springer-Verlag, 2007, pp. 127–137.

F. Geerts, A. Kementsietsidis, and D. Milano, "Mondrian: Annotating and querying databases through colors and blocks," in *Data Engineering, 2006. ICDE '06. Proceedings of the 22nd International Conference on*, april 2006, p. 82.

D. Maier, A. O. Mendelzon, and Y. Sagiv, "Testing implications of data dependencies," *ACM Trans. Database Syst.*, vol. 4, pp. 455–469, December 1979.

J. Delgado and N. Ishii, "Memory-based weighted-majority prediction for recommender systems," in *Proceedings of ACM SIGIR '99 Workshop on Recommender Systems: Algorithms and Evaluation*, ser. ACM SIGIR '99, 1999.

S. Buchegger and J.-Y. L. Boudec, "A robust reputation system for p2p and mobile ad-hoc networks," in *Proceedings of the Second Workshop on the Economics of Peer-to-Peer Systems*, 2004.

A. Whitby, A. Josang, and J. Indulska, "Filtering Out Unfair Ratings in Bayesian Reputation Systems," *The Icfain Journal of Management Research*, vol. 4, no. 2, pp. 48–64, February 2005.

K. Walsh and E. G. Sirer, "Fighting peer-to-peer spam and decoys with object reputation," in *P2PECON '05: ACM wworkshop on Economics of peer-to-peer systems.* NY, USA: ACM Press, 2005, pp. 138–143.

J. Weng, C. Miao, and A. Goh, *Protecting Online Rating Systems from Unfair Ratings*, ser. Trust, Privacy and Security in Digital Business. LNCS. Springer Berlin / Heidelberg, August 2005, vol. 3592, pp. 50–59.

J. Macqueen, "Some methods for classification and analysis of multivariate observations," in *Proc. 5th Berkeley Symp. Mathematical Statist. Probability*, 1967, pp. 281–297.

J. Sonnek and J. Weissman, "A quantitative comparison of reputation systems in the grid," in *Proc. 6th IEEE/ACM International Workshop on Grid Computing*, ser. GRID '05. Washington, DC, USA: IEEE Computer Society, 2005, pp. 242–249.

A. Whitby, A. Josang, and J. Indulska, "Filtering out unfair ratings in bayesian reputation systems," *Science*, vol. 4, no. 2, p. 106117, 2004.

S. Buchegger and J.-Y. Le Boudec, "A robust reputation system for p2p and mobile ad-hoc networks," in *Proceedings of the Second Workshop on the Economics of Peer-to-Peer Systems*, 2004.

ABSTRACT

SEMANTIC WEB BASED RELATIONAL DATABASE ACCESS WITH CONFLICT RESOLUTION.

by

FAYEZ KHAZALAH

August 2015

Advisor: Dr. Zaki Malik

Major: Computer Science

Degree: Doctor of Philosophy

This thesis focuses on (1) accessing relational databases through Semantic Web technologies and (2) resolving conflicts that usually arises when integrating data from heterogeneous source schemas and/or instances.

In the first part of the thesis, we present an approach to access relational databases using Semantic Web technologies. Our approach is built on top of *Ontop* framework for Ontology Based Data Access. It extracts both *Ontop* mappings and an equivalent OWL ontology from an existing database schema. The end users can then access the underlying data source through $SPARQL$ queries. The proposed approach takes into consideration the different relationships between the entities of the database schema when it extracts the mapping and the equivalent ontology. Instead of extracting a flat ontology that is an exact copy of the database schema, it extracts a rich ontology. The extracted ontology can also be used as an intermediary between a domain ontology and the underlying database schema. Our approach covers independent or master entities that do not have foreign references, dependent or detailed entities that have some foreign keys that reference other entities, recursive entities that contain some self references, binary join entities that relate two entities

together, and n-ary join entities that map two or more entities in an n-ary relation. The implementation results indicate that the extracted *Ontop* mappings and ontology are accurate. i.e., end users can query all data (using SPARQL) from the underlying database source in the same way as if they have written *SQL* queries.

In the second part, we present an overview of the conflict resolution approaches in both conventional data integration systems and collaborative data sharing communities. We focus on the latter as it supports the needs of scientific communities for data sharing and collaboration. We first introduce the purpose of the study, and present a brief overview of data integration. Next, we talk about the problem of inconsistent data in conventional integration systems, and we summarize the conflict handling strategies used to handle such inconsistent data. Then we focus on the problem of conflict resolution in collaborative data sharing communities. A collaborative data sharing community is a group of users who agree to share a common database instance, such that all users have access to the shared instance and they can add to, update, and extend this shared instance. We discuss related works that adopt different conflict resolution strategies in the area of collaborative data sharing, and we provide a comparison between them. We find that a Collaborative Data Sharing System (CDSS) can best support the needs of certain communities such as scientific communities. We then discuss some open research opportunities to improve the efficiency and performance of the CDSS. Finally, we summarize our work so far towards achieving these open research directions.

AUTOBIOGRAPHICAL STATEMENT

Fayez Khazalah was born in Mafraq, Jordan. He received his BSc. and MSc. degrees in Computer Science from Al alBayt University in 1999 and 2005, respectively. He joined Wayne State University in the Fall of 2009 to pursue a PhD degree in Computer Science. He also received a second MSc. degree in Computer Science from Wayne State University while he was doing his PhD. Before that he worked for the Customs Department and the Free Zones Corporation in Jordan as a Programmer and System Analyst. His research interests include data integration, semantic web, and ontology-based data access. He is a member of the ACM and IEEE.