International Journal of Semantic Computing
Vol. 4, No. 1 (2010) 123–151

DOI: 10.1142/S1793351X10000961

World Scientific
www.worldscientific.com

ONTOGRATE: TOWARDS AUTOMATIC INTEGRATION FOR RELATIONAL DATABASES AND THE SEMANTIC WEB THROUGH AN ONTOLOGY-BASED FRAMEWORK

DEJING DOU*, HAN QIN† and PAEA LEPENDU‡

Advanced Integration and Mining Lab
Computer and Information Science
University of Oregon, Eugene, OR 97403, USA
**dou@cs.uoregon.edu*
†*qinhan@cs.uoregon.edu*
‡*paea@cs.uoregon.edu*
http://aimlab.cs.uoregon.edu

Integrating existing relational databases with ontology-based systems is among the important research problems for the Semantic Web. We have designed a comprehensive framework called OntoGrate which combines a highly automatic mapping system, a logic inference engine, and several syntax wrappers that inter-operate with consistent semantics to answer ontology-based queries using the data from heterogeneous databases. There are several major contributions of our OntoGrate research: (i) we designed an ontology-based framework that provides a unified semantics for mapping discovery and query translation by transforming database schemas to Semantic Web ontologies; (ii) we developed a highly automatic ontology mapping system which leverages object reconciliation and multi-relational data mining techniques; (iii) we developed an inference-based query translation algorithm and several syntax wrappers which can translate queries and answers between relational databases and the Semantic Web. The testing results of our implemented OntoGrate system in different domains show that the large amount of data in relational databases can be directly utilized for answering Semantic Web queries rather than first converting all relational data into RDF or OWL.

Keywords: Ontology-based information integration; ontology mapping; data mining; relational database; semantic web.

1. Introduction

Many publicly available, structurally and semantically rich resources such as databases, the World Wide Web, and the Semantic Web [11] provide a unique and challenging opportunity to integrate information in new and meaningful ways. From the standpoint of a user trying to integrate data from a variety of sources, the problem of information integration is to provide a unified view that he or she can understand, query, and process independently of the underlying source heterogeneities [28, 32, 33]. While standards such as SQL, XML, and OWL [2] cut down on the *syntactic* diversity, it is unreasonable to expect schemas or ontologies

that describe the *structure* and *semantics* of data to be few in number [14]. As Haas has pointed out [28], information integration remains a challenging problem because various tools and technologies out there currently lack a unified framework that holistically tackles the entire task. The challenge is that these components need to work together and the requirement for human involvement needs to be reduced.

The Semantic Web is an evolving development of the World Wide Web in which the semantics of information and services is formally defined [11]. The main approach is to make the Web content annotated with formal languages and ontologies. Then software agents can take the annotated information for automatic processing and integration. The core of the Semantic Web includes formal specifications such as Resource Description Framework (RDF), RDF Schema (RDFS) and the Web Ontology Language (OWL) [2], all of which are intended to provide a formal description of concepts, terms, and relationships and to enable automatic reasoning (inference) within a given domain.

While we have seen many achievements to realize the Semantic Web by developing ontology languages (e.g., OWL) and ontology-based systems (e.g., OWL reasoners and Semantic Web services [39]), how to apply ontologies and ontology-based systems to integrate and utilize existing real life data resources, especially relational databases, is among the important research problems for the Semantic Web. For example, a Semantic Web agent needs a piece of information (e.g., the middle name of a citizen with some social security number) which is very possibly stored in one or several existing relational databases but the agent is not sure which one. It seems to be an easy problem if the Semantic Web agent is "smart" enough to compose SQL queries to all candidate databases. However, it is more natural that Semantic Web agents compose queries in ontology-based query languages, e.g., OWL-QL [27]. Another alternative is that we can transform all data in candidate relational databases into RDF by annotating data with some ontologies on citizens. Then we can use SPARQL [4] to issue a query based on the RDF model. But it is obviously not efficient because it requires transforming all data into RDF in advance and the information needed is only a very small portion or may not be in any candidate database. It would clearly be better if ontology-based queries could directly retrieve the specific data required via SQL rather than first transforming potentially gigabytes of relational data into RDF unnecessarily.

On the other hand, one single query cannot directly retrieve data from multiple heterogeneous databases. In general, this is an integration problem between relational databases and the Semantic Web but some specific challenges need to be considered. In addition to syntax and structure, there are important semantic differences between Semantic Web ontologies and database schemas. Therefore, the problem is not only a syntax translation between Semantic Web queries and SQL queries. For example, even when describing the same domains, ontologies include more formal semantics than relational models and schemas. Furthermore, the Semantic Web

uses the open-world assumption (OWA) but traditional relational databases use the closed-world assumption (CWA) which makes it difficult and perhaps unnatural to guarantee logical completeness of a combined system.

To address the significant challenges mentioned above, we have designed and implemented OntoGrate, an ontology-based framework towards automatic information integration. This work mainly focuses on the integration of relational databases and the Semantic Web in a highly automatic way which includes an ontology mapping system, an inference engine and several syntax wrappers. We emphasize that the system will be ontology-based because ontologies are more expressive than schemas in describing data semantics. The OntoGrate system can automatically represent a schema as a *DB ontology*. With the generated DB ontology, a semantic web query (e.g., in OWL-QL) can be directly translated into a SQL query and the answers (relational data) can be translated back to semantic web languages (e.g., RDF and OWL). In the ontology matching and mapping process, we apply a variety of state-of-the-art techniques (i.e., string similarity matching, object reconciliation and multi-relational data mining) to get executable mapping rules with high precision and recall. Finally, we have extended an inference engine, OntoEngine [22], for conjunctive query translations with the discovered mapping rules. Our conjunctive query translation is proved to be *sound*.

The rest of this paper is organized as follows. We first introduce some related work in Sec. 2, and then illustrate our OntoGrate framework and the major interactions of its components in Sec. 3. We introduce the formal definition of DB ontology, how to translate a database schema to a *DB ontology*, and how to accomplish query answering in Sec. 4. In Sec. 5, we show how our mapping system discovers mapping rules between heterogeneous DB ontologies through string similarity matching, object reconciliation and multi-relational data mining in a highly automatic way. With discovered mappings, we introduce our inferential query translation process for integrating heterogeneous databases in Sec. 6. Then we test the OntoGrate system through experiments on synthetic and real data from three different domains in Sec. 7. We discuss some interesting observations from the OntoGrate research and future work in Sec. 8. We conclude the paper by summarizing our contributions in Sec. 9.

2. Related Work

The primary goal of this paper is to show how the Semantic Web and relational databases can be integrated in a highly automatic way by combining an ontology mapping system, an inference engine and several syntax wrappers. This is an extended version based on our previous conference and workshop papers [20, 21, 48], each of which shows the progress of some components of OntoGrate. In this paper, we show OntoGrate in a unified semantics by giving formal definitions, algorithms and proofs for our DB ontologies, ontology mapping and query translation.

We also show our recent data experiments from three different domains in terms of the accuracy of mappings and the scalability of query answering. Other related approaches are found in several disciplines:

Database Reverse Engineering and Ontology Extraction: There are a few approaches investigating the transformation of relational schemas to ontologies. One similar approach to our DB ontology generation is to map a relational model to frame logic which can then be represented in RDF [54]. Premerlani [47] proposed a seven-step reverse engineering process and gave the guidelines to get mappings between semantic models and original schemas. A recent research described in [34] provided a description logic based ontology language which captures features from ER and UML class diagrams. It is proven to preserve the semantics of the constraints in the relational databases. Motik, Horrocks and Sattler [41] showed that integrity constraints (ICs) can be disregarded while answering positive queries against an OWL [2] ontology, if the constraints are satisfied by the database. This theoretical research also helped justify that the semantics of databases, including integrity constraints, can be fully preserved in OWL with proper extensions. The focus of our paper is not just on semantic representation for schemas, but more importantly to show how the semantic representations for database schemas (i.e., *DB ontologies*) can help information integration.

Ontology (*schema*) *Matching and Mapping*: We are careful to distinguish between *matching* and *mappings*. What do we mean by this? A *matching* pairs (or groups) related concepts together, whereas a *mapping rule* explicitly states how each element in a pair (or a group) relates to the others in formal languages. The database community was one of the first to invest considerable effort in developing systems that match different database schemas (see [49] for a survey). Most schema matching systems, such as LSD [17], CUPID [35], iMap [16] and COMA [23], focus on retrieving correspondences between attributes using a variety of similarity or correlation heuristics. Clio [29, 40] and Semap [8] are schema mapping systems that can generate operational (i.e., executable) rules in formal languages. Similarly, research in knowledge engineering and the Semantic Web has resulted in tools for ontology matching and mapping. PROMT [43], GLUE [18], MAFRA [36] and BMO [31] are some examples of such systems. BMO [31] can generate block matchings using a hierarchical bipartition algorithm. This system builds a virtual document for each ontology and compares each pair of concepts with the information in the virtual document. GLUE [18] employs machine learning and exploits data instances to find matchings between concepts. A good survey of ontology matching techniques can be found in Euzenat and Shvaiko's book [24]. Matchings can also be used for ontology merging process, such as the approaches in Chimaera [38] and C-OWL [13]. Our ontology mapping approach in OntoGrate is to combine ontology matching, object reconciliation [19] and multi-relational data mining [42] to find executable ontology mapping rules in a highly automatic manner by utilizing the overlapping of data.

Data Integration and Ontology-Based Integration: General data integration models such as federated databases [52], data warehouses [12] and peer-to-peer data management [30] exist. The integration process is normally implemented with a data mediator and an integrated schema represented as view definitions. In data integration and data exchange research [25, 32, 33], the query based on one schema can be answered by rewriting it as the query based on another schema. Most of them are view-based query answering, e.g., Global-As-View (GAV) and Local-As-View (LAV). In the GAV approach, the elements of the global schema are defined as views (i.e., mappings) over the local schemas. In contrast, the elements of the local schemas are defined as views (i.e., mappings) over an existing global schema in the LAV approach. For example, the MiniCon algorithm [46] rewrites queries expressed in a global view to a conjunction of queries over local ones so that a large set of (materialized) views can lead to efficient query answering. The query rewriting is defined by *query containment* [33]. Ontology-based information integration approaches based on a declarative model (as opposed to a procedural one) often use a logical framework from the area of knowledge representation. The Carnot, SIMS and Information Manifold systems are summarized and compared in [15]. Poggi *et al.* [45] presented how to design an effective system for ontology-based data access. Pierre [44] discussed the difference between applying a conceptual model and ontology to data integration and argued that ontology is more suitable. Our query translation from an ontology to database schemas is defined by logic entailment and implemented as a sound inference.

3. Framework

We have developed OntoGrate, a framework that, given different ontologies or schemas and their associated data, will be able to mine a set of first-order mapping rules that accurately describe how the input ontologies or schemas relate to each other. These mapping rules will be used by an inference engine, OntoEngine, to perform data integration (e.g., query translation). For example, in order to find the middle name of a citizen with a specified social security number as a global query, the Semantic Web agent needs to identify the sources,[a] translate that query to each source using inference, retrieve the data with wrappers, and return the reconciled answers. It hides several tasks that need to be taken care of: we need a model of how the global query is related to each source (i.e., mapping discovery) and a way to utilize those mappings to translate the query in the expected way. Finally we need to execute each query against each local data resource using syntax wrappers. To explain how OntoGrate can help integrate databases and the Semantic Web, we show the architecture of OntoGrate in Fig. 1.

[a]Semantic search is an important research topic but not the focus of this paper. In this paper, we assume that the Semantic Web agent has already identified the candidate databases or Semantic Web documents to answer the query.

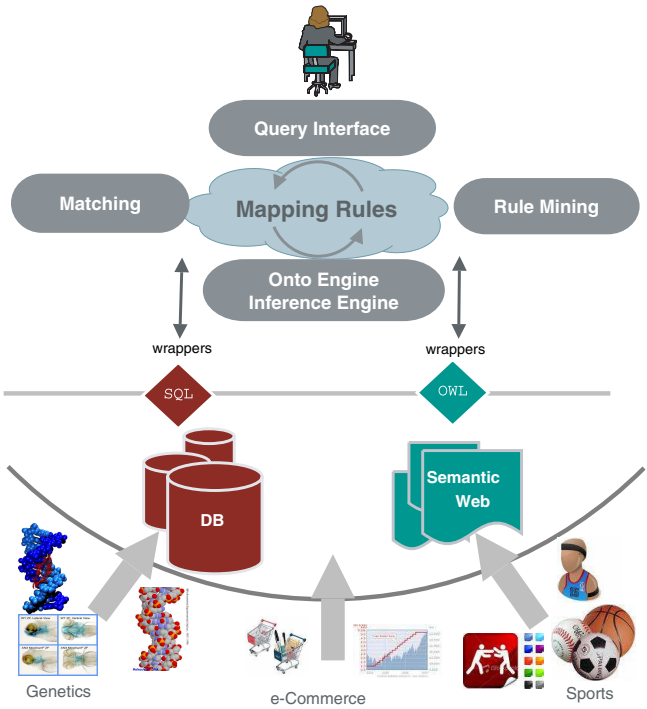

Fig. 1. **Architecture for OntoGrate.** The system is mainly composed of five components: the *ontology matching*, the *rule miner*, the *inference engine*, the *query interface*, and *syntax wrappers*. The *transformation from schema to ontology* is implemented in the wrappers between SQL and OWL. The inputs are relational databases and Semantic Web documents and queries over heterogeneous schemas or ontologies from various domains.

Schemas to Ontologies Transformation: We have built an automatic translator inside the wrappers between SQL and OWL to represent schemas as ontologies. Some basic heuristics have been used for transforming schemas into ontologies which we call *DB ontologies*.

Ontology Matching: Ontology matching takes the source and target ontologies with their data instances as input. The output is 1-1 matching pairs between classes and properties. OntoGrate also reconciles instances which refer to the same real-world entities.

Mapping Rule Mining: OntoGrate first combines closely related 1-1 property matching pairs, class matching pairs and their instances together as a group. Then the matching groups and associated data are input to a FARMER-like system [42] to mine *frequent queries* which help to generate the final mapping rules.

Inference Engine: The inference engine is in charge of using mapping rules to answer queries and exchange (i.e., translate) data among available data sources.

OntoEngine [22] has been extended for conjunctive query translation. We will provide the theory and proof in Sec. 6.

Query Interface: In this paper, we assume users (i.e., human or semantic web agents) issue queries in semantic web query languages (e.g., OWL-QL or SPARQL). The final query answering results can be represented in the query interface.

Syntax Translation (Wrappers): In between OntoGrate and the data resources exist syntax translators (wrappers) among OWL, SQL, OWL-QL and our language Web-PDDL [37]. The query wrapper should take a fully-translated ontology-based query and efficiently generate the corresponding data-access SQL query without additional translation or rewriting cost. Until the proposed standards for semantic mappings (open question [55]) are finalized, Web-PDDL is used internally to describe both the structure and semantics of data resources, their mappings and queries.

4. Transforming Relational Schemas to DB Ontologies

4.1. *Problem definition*

The structure and integrity constraints of relational tables are defined by relational models and database schemas (e.g., definitions with SQL). When Semantic Web agents (which use ontologies) want to interact with relational databases, we need to deal with both the semantic differences between ontologies and schemas and the syntax differences (e.g., OWL vs. SQL). A relational schema is a finite set $\mathbf{R} = \langle R_1, R_2, \ldots, R_n \rangle$ of relations, each of which may have different arities. On the other hand, Semantic Web ontologies (i.e., OWL ontologies) use description logic (i.e., a decidable fragment of first-order logic) as their logic foundation. OWL ontologies mainly have classes, binary predicates (properties) and some axioms (such as cardinality constraints). Our idea is to automatically create a Semantic Web ontology which can describe the semantics and structure defined by a database schema, then Semantic Web agents can query the corresponding database based on that Semantic Web ontology.

To do so, we first borrow the ideas from both Datalog representation and Reiter's first-order database [50] to redefine relational databases in more general first-order logic (FOL) theory. More detailed discussion between database and logic can be found in Atzeni and De Antonellis's book [9].

Definition 1. Relational Database: A relational database DB is a finite set of relations $R_1, R_2, \ldots, R_n$, each of which can be defined as a predicate (literal) $R_i(x_1, x_2, \ldots, x_{m_i})$, where $1 \leq i \leq n$, $m_i \geq 1$ and m_i is the number of attributes in R_i. Variable x_j $(1 \leq j \leq m_i)$ corresponds to an attribute with the data type of D_j in the relation R_i. The data type D_j can also be defined as a unary predicate $D_j(x_j)$. If $c_1, c_2, \ldots, c_{m_i}$ are constants, then $(c_1, c_2, \ldots, c_{m_i})$ is an instance of R_i if and only if $\mathrm{DB} \vdash R_i(c_1, c_2, \ldots, c_{m_i})$, where $\vdash$ means the instance of R_i can be logically inferred from DB.

Most integrity constraints of a relational database can be expressed by first-order logic formulas (axioms) [10]. A conjunctive database query is basically a conjunction of predicates defined from data types and relations and comparison operators:

$$q(\vec{X}) \leftarrow \exists\, \vec{Y} conj\,(\vec{X}, \vec{Y}, \vec{Z})$$

where $\vec{X}$ represents distinguished variables which need to be bound to answers in the DB, $\vec{Y}$ represents non-distinguished variables as existential variables and $\vec{Z}$ can be constants. And $conj(\vec{X}, \vec{Y}, \vec{Z})$ is a conjunction of atoms of the form $D(x)$, $R(x_1, \ldots, x_n)$, or $C(t_1, t_2)$, where D, R, C are data types, predicates and comparison operators respectively. $x, x_1, \ldots, x_n$, t_1 and t_2 are variables in X and Y or constants in Z. An n-tuple of constants $\mathbf{c} = (c_1, \ldots, c_n)$ is an answer to $q(\vec{X})$ (with respect to a database DB) if and only if $DB \vdash D_1(c_1) \wedge \cdots \wedge D_n(c_n) \wedge \theta(q(\vec{X}/\mathbf{c}))$ where $\vec{X}$ has n variables and $\theta(q(\vec{X}/\mathbf{c})$ is the ground formula achieved by substituting the variables in $\vec{X}$ with the corresponding constants in $\mathbf{c}$. An answer sequence is a list of answers to the query.

We then use a similar way defining relational databases to define a Semantic Web ontology for relational databases. To be consistent with OWL ontologies which contain mainly classes (unary predicates) and properties (binary predicates), we give the following definition for a *DB ontology*:

Definition 2. DB Ontology: A DB ontology can be defined as a finite set of classes (i.e., types or unary predicates) and binary predicates. If R_1,$R_2, \ldots, R_n$ are the relations in a database DB, we define them as classes in the corresponding DB ontology. Each attribute a_j with data type of D_j in an original relation R_i ($1 \leq j \leq m_i$, m_i is the number of attributes in R_i) can be defined as a binary predicate $a_j(r_i, d_j)$, where r_i and d_j are variables with types (classes) of R_i and D_j. If c_1, $c_2, \ldots, c_{m_i}$ are constants, then there exists an r_0 and $a_1\ (r_0, c_1) \wedge a_2\ (r_0, c_2) \wedge \cdots \wedge a_{m_i}(r_0, c_{m_i})$ is an instance of $a_1 \wedge a_2 \wedge \cdots \wedge a_{m_i}$ if and only if DB $\vdash a_1\ (r_0, c_1) \wedge a_2\ (r_0, c_2) \wedge \cdots \wedge a_{m_i}(r_0, c_{m_i})$, where r_0 is a Skolem term (with type of R_i) based on $c_1, c_2, \ldots, c_{m_i}$ and $\vdash$ means the instances of $a_1, \ldots, a_{m_i}$ can be logically inferred from DB.

However, the definitions of unary or binary predicates in DB ontologies cannot catch all semantics of databases. For example, from the data modeling point of view (e.g., ER model), it is very possible some relation is designed based on one relationship between two entities in the original ER model. We may not need to define this kind of relation again as classes (unary predicate), instead we can define it as a binary predicate between two classes (i.e., objectProperty in OWL). The primary key constraints can be represented as functional property axioms and other integrity constraints (e.g., foreign keys and other available functional dependencies) of a relational database can also be expressed by formulas (axioms) in first-order logic (more detail examples in Sec. 4.2). Since the classes can be viewed as unary predicates, a query over DB ontology is also a form of conjunction of predicates (which includes unary and binary predicates and comparison operators) with

variables and constants. Therefore, a query over DB ontology is still in the form of

$$q(\vec{X}) \leftarrow \exists\, \vec{Y} conj(\vec{X}, \vec{Y}, \vec{Z})$$

where $\vec{X}$ represents distinguished variables which need to be bound to answers in the DB, $\vec{Y}$ represents non-distinguished variables as existential variables and $\vec{Z}$ can be constants. And $conj(\vec{X}, \vec{Y}, \vec{Z})$ is a conjunction of atoms of the form $D(d)$, $R(r)$, $a(r, d)$ or $C(t_1, t_2)$, where D, R, a, C are respectively data types, relation types, properties (binary predicates) and comparison operators. Also, r, d, t_1 and t_2 are variables in X and Y or constants in Z. An n-tuple constant $\mathbf{c} = (c_1, \ldots, c_n)$ is an answer to $q(\vec{X})$ only if there exist Skolem terms $r_1, \ldots, r_m$ based on $(c_1, \ldots, c_n)$ that

$$DB \vdash D_1(c_1) \wedge \cdots \wedge D_n(c_n) \wedge R_1(r_1) \wedge \cdots \wedge R_m(r_m) \wedge \theta(q(\vec{X}/\mathbf{c}))$$

where $\vec{X}$ has n variables and $\theta(q(\vec{X}/\mathbf{c})$ is the ground formula (e.g., $a_1(r_1, c_1)$) achieved by substituting the variables in $\vec{X}$ with the corresponding constants in $\mathbf{c}$. An answer sequence is a list of answers to the query.

4.2. *Transforming a DB schema to a DB ontology*

After we define both the relational database (schema) and DB ontology in first-order logic theories, we list some general correspondences (i.e., heuristics) between a relational database and its DB ontology:

$$\begin{aligned} \text{Relation} &\rightsquigarrow \text{Class (Type)} \\ \text{Attribute} &\rightsquigarrow \text{Property} \\ \text{Integrity Constraint} &\rightsquigarrow \text{Axiom (Rule)} \\ \text{Primary Key} &\rightsquigarrow \text{Functional Property} \end{aligned}$$

We have developed an automatic translator which utilizes the above heuristics and includes syntax wrappers to generate a DB ontology from a database schema. The translation is between the SQL data-definition language (DDL) and our internal first-order Web ontology language, Web-PDDL [37], to express ontologies, data instances (facts), queries, and mapping rules between different ontologies. The syntax wrapper between SQL queries and Web-PDDL queries is called PDDSQL [21]. Web-PDDL can be translated to and from RDF easily [37]. Another syntax wrapper called PDDOWL [21] can translate OWL and OWL-QL to and from Web-PDDL as well. Therefore, DB ontologies can be represented in OWL and a query over a DB ontology also can be represented as an OWL-QL query. To make the syntax as consistent as possible, in this paper we use general first-order logic formulas to represent our internal Web-PDDL representations instead of Lisp syntax.

To illustrate our approach, we consider an IBM Informix[b] database schema, Stores7, from the online sales domain, which we installed and populated data for

[b] http://www.ibm.com/software/data/informix/.

the experiments. The database schema of Stores7 defines the relations, such as `Customer`, `Order` and `Item`, and associated attributes.

Our automatic tool can translate this relational schema (in SQL DDL) into a DB ontology. For example, there is a `Customer` table in Stores7, where `customernumber` is the primary key of the table and `customerstatecode` is the foreign key that refers to the `State` table. Our tool generates the first-order logic formulas:

$$
\begin{aligned}
&Classes : Customer(x), State(y) \ldots \\
&Subclasses : \forall\, x\ Customer(x) \rightarrow Relation(x), \forall\, y\ State(y) \rightarrow Relation(y) \ldots \\
&Predicates : customernumber(x, v), statecode(y, v), customercity(x, v) \ldots \\
&Axioms : \forall\, c, o\ Customer(c) \wedge varchar(o) \wedge customerstatecode(c, o) \\
&\qquad \rightarrow \exists s\ State(s) \wedge\ statecode(s, o) \\
&\qquad\qquad \forall\, c1, c2, n\ Customer(c1) \wedge Customer(c2) \wedge String(n) \wedge\ customernumber(c1, n) \\
&\qquad\qquad\qquad \wedge\ customernumber(c2, n) \rightarrow\ (c1 = c2) \\
&\qquad\qquad \ldots
\end{aligned}
$$

The above representation is part of the DB ontology of the Stores7 database, which specifies the *Customer* and *State* classes and their related predicates (properties) and axioms based on foreign keys and primary keys (e.g., *customerstatecode* and *customernumber*). For example, the `Customer` relation (table) can be represented as a *Customer* class (unary predicate) and also a subclass of *Relation* which is a super class for all classes transformed from relation tables. Nine attributes of *Customer* table can be represented as nine binary predicates (e.g., *customercity*).

To make the DB ontology "understandable" by the Semantic Web agents, we use OWL to represent classes (types) and binary predicates, and as many axioms (integrity constraints) as OWL can. For example, the primary key information can be represented as functional property, e.g., *customernumber* can be represented as a functional property related to *Customer* class (i.e., domain) and a string (i.e., range). We may use the current SWRL [5] or RIF [3] rule language to represent other more general axioms (e.g., foreign key constraints) but we generally use first-order logic syntax to represent rules in this paper. After we get the Stores7 DB ontology in Web-PDDL, we call our syntax wrapper PDDOWL to transform it into OWL syntax like:

```
 <owl:Class rdf:ID="Customer">
  <rdfs:subClassOf rdf:resource="&sql;Relation"/>
 </owl:Class>
 <owl:DatatypeProperty rdf:ID="customernumber">
  <rdf:type    rdf:resource="&owl;FunctionalProperty" />
  <rdfs:domain rdf:resource="#Customer" />
  <rdfs:range rdf:resource="&xsd;String"/>
</owl:DatatypeProperty>
 <owl:DatatypeProperty rdf:ID="customercity">
  <rdfs:domain rdf:resource="#Customer"/>
  <rdfs:range rdf:resource="&xsd;String"/>
 </owl:DatatypeProperty>
...
```

Suppose Semantic Web agents or services may want to use OWL-QL (or SPARQL) to query databases with the DB OWL ontology. PDDOWL first translates OWL-QL queries into Web-PDDL queries. Then we call the PDDSQL translator to translate Web-PDDL queries into SQL queries by using the general correspondences between a relational schema and a DB ontology. Finally we can get answers (i.e., data tuples) directly from the relational database. The answers can be filled in to provide answer bundles in OWL-QL by PDDOWL. We will give the detailed examples in Sec. 7.

5. Ontology Mapping with Data Mining

Although we represent the semantics of relational databases as DB ontologies, different databases may result in different DB ontologies even in the same domain if their schemas are heterogeneous. One advantage of OntoGrate is to be able to discover semantic mappings of heterogeneous databases as an ontology mapping process in a highly automatic way.

Definition 3. Ontology Mapping: Let $\vec{S}$ and $\vec{T}$ be the source and target ontology respectively. The ontology mapping between $\vec{S}$ and $\vec{T}$ is a triple $\mathcal{M} = (\vec{S}, \vec{T}, \Sigma)$ such that Σ is a set of mapping rules as formulas of some logic $\mathcal{L}$ (e.g., FOL).

We mentioned the difference between matchings and mapping rules in Sec. 2. In general, given a source ontology and a target ontology which model the same domain, ontology matching finds the correspondences of concepts (e.g., classes and properties in OWL ontologies). Object reconciliation can find that some data instances from both ontologies represent the same real-world entities. The goal of ontology mapping is to generate mapping rules which can be represented in some formal logic. Especially, mapping rules discovered by our system can be represented as implications with conjunctions:

$$\forall\, x_1 \ldots x_k,\ P_1 \wedge \cdots \wedge P_i \cdots \wedge P_n \rightarrow \exists\, z_1 \ldots z_l, Q_1 \wedge \cdots \wedge Q_j \cdots \wedge Q_m$$

where the x's are universally quantified variables and z's are existentially quantified variables. $P_1, \ldots, P_n$ are unary or binary predicates from the source ontology and $Q_1, \ldots, Q_m$ are unary or binary predicates from the target ontology.

There are mainly four steps to do ontology mapping in OntoGrate:

(1) Matching Generation: It takes the source and target ontologies with their data instances as input. The output is 1-1 matchings between classes and properties.
(2) Object Reconciliation: This step reconciles instances which refer to the same real-world entities. It iteratively takes matchings as input and outputs reconciled data instances to the matching generator as feedback. When no new matching is generated, it passes the 1-1 matchings and the reconciled data to the Group Generation.

(3) Group Generation: The matchings are not always 1-1 matchings. This step combines closely related 1-1 property matchings, class matchings and their instances together as a group which may be m-n matchings.
(4) Multi-Relation Data Mining and Rule Refining: We extend some ideas from the FARMER system [42] to mine *frequent queries* which can help generate executable mapping rules. The thresholds of support and confidence for rule selection can be different for different cases.

In the following, we illustrate our ontology mapping approach based on two sales DB ontologies: the Stores7 Ontology we mentioned in previous sections and the Northwind Ontology which is transformed from the Northwind sales database from Microsoft. We use "@stores7" and "@nwind" to denote their namespaces.

5.1. *Matching generation and object reconciliation*

We begin from finding class and property matching pairs based on their names. We make use of "Iterative SubString Matching Algorithm" [53] to calculate the similarity of names of each pair. We basically examine every pair of classes and properties from both DB ontologies. For each pair we can get a similarity score and we set a threshold for similarity to get class or property matching pairs. Other existing matching approaches (e.g., synonym-based approaches by using WordNet [26]) can also be used in this step to help find more matchings.

5.1.1. *Class and property matching*

There are some property matching pairs which can strengthen our confidence about potential class matching pairs. Given the datatype property pair $p(X, Y)$ and $q(U, V)$, where X and U are classes and Y and V are data types, if there is a class matching pair $X \leftrightsquigarrow U$ and Y being the same data type as V, we consider $p(X, Y) \leftrightsquigarrow q(U, V)$ as one potential datatype property matching related to the class matching pair $X \leftrightsquigarrow U$. Not only is the name similarity of p and q needed to make more confidence of $X \leftrightsquigarrow U$, but also the data value similarity of p and q. The potential datatype property matching will be verified by data value similarity which will be further discussed in Sec. 5.1.2. The higher the similarity of a datatype property matching pair is, the more confidence this class matching pair has. Support of a class matching pair is calculated according to the following equation:

$$Support(X \leftrightsquigarrow U) = \Sigma DatatypePropertyPairSimilarity \tag{1}$$

Datatype property pair similarity is the sum of name similarity and data similarity. Another problem we should cope with is that two classes may have totally unrelated names, but they represent the same concept. One clue to handle this case is actually from datatype property matchings. If several property matching pairs indicate that class X and class U should be paired, we can assume $X \leftrightsquigarrow U$ is one class matching pair and add those property matching pairs as its datatype property matchings.

5.1.2. *Data similarity of datatype property pairs*

Having a similar name does not necessarily mean that two properties definitely represent the similar concept. Calculating data similarity of property matching pairs is necessary. Note that data similarity is based on the assumption that the data instances of two ontologies overlap at a relatively high level, otherwise we cannot benefit from making use of the data. For a property matching pair $p(X,Y) \leftrightsquigarrow q(U,V)$, we can calculate the data similarity of p and q with the following formula:

$$DataSimilarity\,(p(X,Y) \leftrightsquigarrow q(U,V)) = \frac{2 * |same\_pairs(Y,V)|}{|p(X,Y)| + |q(U,V)|} \tag{2}$$

where $|same\_pairs(Y,V)|$ is the number of instance pairs which have the same numeric value or very similar string value, and $|p(X,Y)|$ and $|q(U,V)|$ stand for the numbers of instances of property $p(X,Y)$ and property $q(U,V)$. We give a weight 0.75 to data similarity and 0.25 to name similarity and then set a threshold 0.5 for the overall similarity. If overall similarity of one property matching is less than the threshold, we consider it incorrect and remove it from the list of matching pairs.

5.1.3. *Object reconciliation*

With selected class matching pairs, we adopt an object reconciliation algorithm developed by Dong, Halevy and Madhavan in [19] to reconcile the instances of classes from two ontologies. The original algorithm considers the relationship of matching pairs and determines whether two data objects from databases represent the same real-world entity. We successfully use the idea in our mapping research.

We skip repeating the algorithm in detail since it can be found in the paper [19]. In summary, for all the possible instance pairs, we can draw a similarity graph and calculate the similarity between them. The formula we use is a simple aggregation of the similarity of datatype matching pairs. For example, Fig. 2 shows one positive example and one negative example for object reconciliation for Stores7 and Nothwind. Node (*customer1, cust001*) has a high similarity and is considered as reconciled. Node (*customer2, cust003*) has a comparably low similarity and is considered as not reconciled.

5.1.4. *Object property matching*

Similar to datatype property matching pairs, the object property matching pairs may have similar property names. However, it is harder to find object property matchings because their instances cannot help before object reconciliation process is performed. Therefore, only after we reconcile some data instances we then can calculate the data similarity of object property matching pairs. Note that this kind of matching pairs have two ways to match: given $p(X,Y) \leftrightsquigarrow q(U,V)$, the first matching is $X \leftrightsquigarrow U, Y \leftrightsquigarrow V$ while the other is $X \leftrightsquigarrow V, Y \leftrightsquigarrow U$. When we calculate the

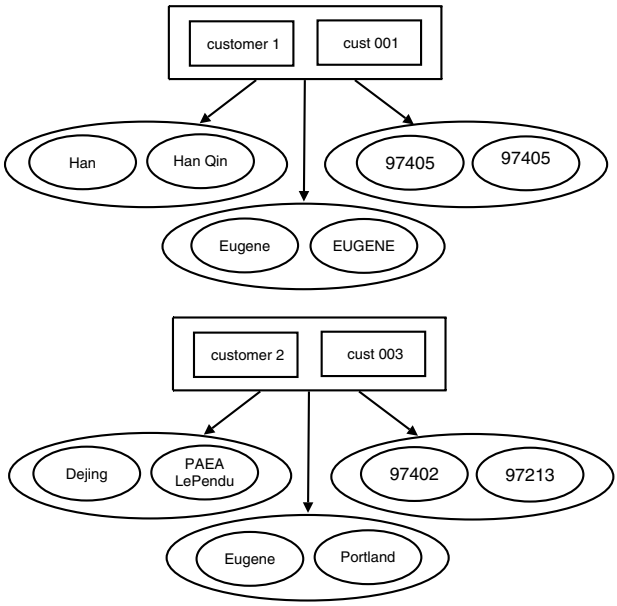

Fig. 2. Positive and negative object reconciliation examples.

data similarity of object property matchings, whether it is cross matched or not should be recorded. Similar to data property matching pairs, we give the following formula:

$$DataSimilarity\,(p(X,Y) \leftrightsquigarrow q(U,V)) = \frac{|same\_pairs(X,U)| + |same\_pairs(Y,V)|}{|p(X,Y)| + |q(U,V)|} \tag{3}$$

After performing object reconciliation in each iteration, our system tries to create new object property matching pairs based on the object reconciliation results. The process will terminate if no new object property matchings can be found.

5.2. *Matching groups*

In this step, we tackle the problem of grouping related 1-1 matchings. For one property matching pair @source:p(X,Y) $\leftrightsquigarrow$ @target:q(U,V), we should find connections between both *@source:X* $\leftrightsquigarrow$ *@target:U* and *@source:Y* $\leftrightsquigarrow$ *@target:V*. Figure 3 shows a complete group.

The dashed line refers to zero or several predicates, super-sub class relationships or class matching pairs. Based on this, we can draw the general group rule: *For one property matching pair, such as @source*: $p(X,Y) \leftrightsquigarrow$ *@target*:$q(U,V)$, *if we do not have class matching pair @source*:$X \leftrightsquigarrow$ *@target*:U (*or @source*:$Y \leftrightsquigarrow$ *@target*:V), *we can search among the properties and class matching pairs to find a connection path from @source*:X *to @target*:U (*or from @source*:Y *to @target*:V). *In the*

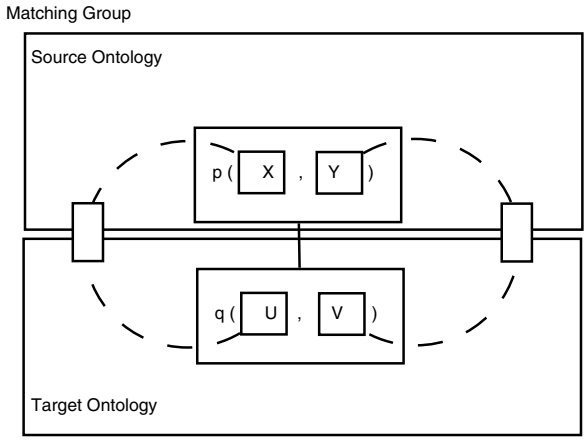

Fig. 3. Matching group based on one property matching pair.

path there must exist one class matching pair that connects the source and target ontologies.

Discovering the path is the key step for finding group matchings. The class matching pair that connects the source and target ontologies is the key class matching pair. We search the class matching pair that could connect the source ontology class and the target ontology class. The properties and superclass-subclass relationships are added into class sets by breadth-first search. Once we find a class (e.g., A) in the source ontology and a class (e.g., B) in the target, which are included in a class matching pair, this $A \leftrightsquigarrow B$ is considered as a key class matching pair and added to the key matching pair set. Then the concepts included in paths will be considered together as a matching group.

5.3. *Generating mapping rules with MRDM*

Once we have all matching groups, the mapping rules can be generated by multi-relational data mining (MRDM). We first introduce some notions for the rule generation with MRDM. Most MRDM systems leverage inductive logic programming (ILP) techniques. In ILP systems, an *atom* is a unary or binary predicate and a *query* is a logical expression that contains a set of ordered atoms. In each query there is one atom called *atom key*, which is used for counting. For one query, if the number of answers of the atom key exceeds the threshold, we refer to this query as *frequent query*. The algorithm should take a set of predicates and data instances as input, build the search space and finally output a set of queries with high support. We consider classes and properties of ontologies as unary or binary predicates. FARMER [42] system can be used for this goal but it requires users to specify the input/output type of each argument of the predicates. To make the whole process as automatic as possible we create a new method based on FARMER. The method

will generate two queries. How many answers those queries have are based on the given data set. We set the number of answers as the support of each query. The first query contains the predicates of the source ontology in the matching group. The second query contains all the predicates of the matching group. The motivation of generating these two queries is that the first query is the condition part of the potential mapping rules and the second query contains both the condition and conclusion parts. The support number will be used for calculating the confidence of the final mapping rules.

Then we can generate mapping rules based on frequent queries and class matching pairs. A query is represented as the logic form $? - P_1, P_2, \ldots, P_N$, which contains an atom key used for counting. For example, the related frequent queries of the matching group "*@stores7:zip* (*Customer*, *String*) $\leftrightsquigarrow$ *@nwind:postalcode* (*Customer*, *String*), *@stores7:Customer* $\leftrightsquigarrow$ *@nwind*: *Customer*" are:

```
? - Customer(V0N0),zip(V0N0,V1N0) support: 100
? - Customer(V0N0),zip(V0N0,V1N0),postalcode(V0N0,V1N0) support: 95
```

where support 100 or 95 means the number of answers for the query from a given dataset. With class matching pair *@stores7:Customer* $\leftrightsquigarrow$ *@nwind:Customer*, a rule can be generated like:

$$\forall x, y \ @stores7{:}Customer(x) \wedge @stores7{:}zip(x, y) \rightarrow @nwind{:}Customer(x) \wedge @nwind{:}postalcode(x, y)$$

The confidence of the mapping rule can be calculated by dividing the support of second query by the support of the first query. In the above example, the confidence of the mapping rule is $\frac{95}{100} = 95\%$. We consider rules with extremely low support and confidence as distinctly incorrect.

6. Integrating Relational Databases with DB Ontologies

In this section, we introduce how to integrate multiple relational databases with discovered mappings and a merged ontology.

6.1. *GAV-like ontology-based integration*

Similar to the creation of a global schema in the GAV data integration, we create a merged ontology based on discovered mappings. Following the ontology merging methodology we used in [22], we put the union of concepts of two DB ontologies into one merged ontology (e.g., Stores7-Northwind) and add the discovered mappings between them as *bridging axioms*. When two DB ontologies have the classes or properties for the same or similar concepts, we choose the concepts from one of (local) DB ontologies and use mappings to specify the relationships between those concepts in the merged ontology and (local) DB ontologies. The process is automatic based on the discovered mapping rules.

We can also represent classes and predicates of the merged DB ontology (e.g., Stores7-Northwind) in OWL. If a query is issued based on the merged DB ontology, the query can be translated into queries in the local DB ontologies by *inferential query translation* (more detail later). Then the translated SQL queries by PDDSQL can be answered by the local databases.

We solve the query answering problem by translating the query from a target ontology (i.e., the merged one) to a source (local) ontology. It may be confusing to think of the merged (global) ontology, the one against which the user's query is posed, as the "target" ontology. Because translation happens using the rules mapped from the *source* DB ontology to the *target* merged ontology (more on this in Sec. 6.2) we like to refer to the merged ontology as the "target" for the sake of consistency. Therefore, our approach is to translate (or rewrite) the query q_t over the target ontology $\vec{T}$ (i.e., the merged one) to a query q_s over the source DB ontology. We call the process *query translation* and formally define it below:

Definition 4. Query Translation: Let $\mathcal{M}= (\vec{S}, \vec{T}, \Sigma)$ be an ontology mapping and let q_t be a query over the target ontology $\vec{T}$. If there is a query q_s over the source DB ontology $\vec{S}$ and the answers to q_s from the DB can be used to answer q_t, we call q_s the query translation of q_t with respect to $\mathcal{M}$.

The advantage here is that we do not need to know which instances are good for answering the query, before we run the query translation. It makes the complexity of query translation controllable independently of the instance-size.

6.2. *Inferential query translation*

If the discovered mapping rules are in first-order logic (e.g., Web-PDDL) and let $\mathcal{M} = (\vec{S}, \vec{T}, \Sigma)$ be an ontology mapping related to the source DB ontology $\vec{S}$ and the target ontology $\vec{T}$, then Σ is the set of mapping rules (first-order axioms). Let the symbol $\rightsquigarrow_Q$ indicate query translation. If q_t is a query in ontology $\vec{T}$, its translation is a query q_s in DB ontology $\vec{S}$, such that any answer (set of bindings) to q_s is also an answer to q_t. In other words:

$$(\Sigma; q_t) \rightsquigarrow_Q q_s \textit{ only if } (\Sigma; \theta(q_s)) \models \theta(q_t)$$

for any substitution θ, where $\theta(q_s)$ is the answer(s) from the source database by substituting the variables in q_s with the corresponding constants in θ. It means we use entailment ($\models$) to define the query translation: if all the mapping rules in Σ and all the answers in $\theta(q_s)$ are true, then all the answers in $\theta(q_t)$ should be true. Alternatively, we say that $\theta(q_t)$ is a logical (or semantic) consequence of Σ and $\theta(q_s)$. It is easy to get in first-order logic theory, for any substitution θ,

$$\begin{aligned}(\Sigma; q_t) \rightsquigarrow_Q q_s \;&\Leftrightarrow\; (\Sigma; \theta(q_s)) \vdash \theta(q_t) \\ &\Rightarrow\; (\Sigma; \theta(q_s)) \models \theta(q_t)\end{aligned}$$

where an (sound) inference ($\vdash$) can implement and guarantee the entailment ($\models$).

Definition 5. Soundness of Query Translation: Let $\vec{T}$ and $\vec{S}$ be the target ontology and the source ontology respectively. Let q_t be a query over $\vec{T}$, and q_s a translation (rewriting) of q_t over $\vec{S}$. The translation (rewriting) process is sound if any answer to q_s is also an answer to q_t.

In the literature this is also known as *query containment* in data integration [33] and *weak representation* in incomplete databases [7].

On the other hand, considering that the Semantic Web adopts the OWA assumption, the answers from one or even all existing databases is not a complete specification of all data. Finding all the (*complete*) answers for a target query is not always possible or necessary. Therefore, we claim that the answers from the source database may not be complete but are correct (sound) with respect to the source query and the ontology mapping $\mathcal{M}$ as long as the inference is sound (a proof later).

In order to use ontology mappings for inferential query translation, the special purpose inference engine OntoEngine [22] was used. OntoEngine has both forward-chaining and backward-chaining reasoners using generalized modus ponens [51], which is well known as a sound but not complete inference algorithm.

The recursive algorithm for conjunctive query translation with backward-chaining inference is shown in Algorithm 1. Basically this IQT algorithm takes a conjunctive query Q_T in the target ontology and the mapping rules Σ between the source and target as input. For each single subquery Q_t, the algorithm translates it to a query Q_s in the target ontology by backward-chaining with *modus ponens*. Q_s can be a single query or a conjunctive query. We combine the Q_ss generated from different subqueries (Q_ts) with conjunctions.

In the following, we give a proof sketch that the conjunctive query translation with backward-chaining can guarantee the soundness of query translation.

Theorem. *Any query translation by Algorithm 1 (IQT) is sound: any answer of translated query is also the answer to the original query.*

Proof. Let q_t be the query over the target ontology $\vec{T}$ and q_s be q_t's translation as a query over the source ontology $\vec{S}$. Based on the Algorithm IQT, there exists a chain (i.e., inference proof) in the form: $\langle q_0, a_0, q_1\rangle$, $\langle q_1, a_1, q_2\rangle, \ldots, \langle q_{n-2}, a_{n-1}, q_{n-1}\rangle$, $\langle q_{n-1}, a_{n-1}, q_n\rangle$, where n is the number of steps to run the modus ponens procedure in the algorithm. To be consistent, we use q_0 to represent q_t, and use q_n to represent q_s. Therefore, for the ith $(0 \leq i \leq n-1)$ step, a_i is the set of axioms that are used to translate q_i to q_{i+1} by the modus ponens procedure.

Let θ_{one} be an answer to q_s as one substitution (bindings) for the variables of q_s. If we substitute all variables of q_s (i.e., q_n), we must get a conjunction of facts (grounded logic formulas) since q_s is a conjunctive query. We name the conjunction of facts as f_s. Using modes ponens in a forward-chaining way [51], we get $(f_s, a_{n-1}) \vdash f_{n-1}$, where f_{n-1} is a conjunction of facts by substituting the variables in q_{n-1} with θ_{one}. Similarly, we can run the forward-chaining from f_{n-1} to f_{n-2}, ... until

Algorithm 1 Inferential Query Translation (IQT)

Input: Conjunctive query Q_T in the target ontology. The mapping rules Σ between the source and target.

Output: Conjunctive query in the source ontology Q_S

Q_S = True
while $\exists$ next subquery Q_T **do**
Query Q_t = next subquery in Q_T
Query Q_s = BackwardChaining (Q_t)
$Q_S = Q_S \wedge Q_s$
end while
Return Q_S

Function BackwardChaining (Query Q)
Query $Q_r = Q$
if Q's predicate is in the target ontology **then**
$Q_r = Q$
else
while $\exists\, m\ (\forall x_1 \ldots x_k,\ P_1 \wedge \cdots \wedge P_i \cdots \wedge P_n \rightarrow \exists z_1 \ldots z_l, Q_1 \wedge \cdots \wedge Q_j \cdots \wedge Q_m)$ in Σ, Q's predicate is same as Q_j in m **do**
New Query Q_N = ModusPonens(Q, m)
Q_r = IQT with the input of Q_N and Σ
end while
end if
Return Q_r

Function ModusPonens (Query Q, Mapping m)
Query $Q_r = Q$
Substitutions = { }
if $Q = Q_j(?x_j, ?y_j)$ and one predicate in the conclusion of m is $Q_j(x_j, y_j)$ **then**
Substitutions = *Substitutions* + $\{x_j/?x_j, y_j/?y_j\}$
end if
if *Substitutions* is not empty **then**
Q_r = Substitute the variables in the condition (i.e., $P_1 \wedge \cdots \wedge P_i \cdots \wedge P_n$) of m according to *Substitutions*.
end if
Return Q_r

f_0 which is a conjunction of facts by substituting the variables in q_t (i.e., q_0) with θ_{one}. The forward-chaining with modus ponens guarantees that f_{n-1}, f_{n-2}, ..., f_0 are all true facts as long as f_s is true (i.e., θ_{one} is an answer to q_s). Therefore, θ_{one} is also an answer of q_t. □

7. Experiments

To test the performance of OntoGrate we have conducted three case studies in different domains. We have tested the scalability of query answering only for the first case because the scalability should be domain independent and the latter two cases have small data sets. For all three cases we also have tested the accuracy of the mapping discovering.

7.1. *Sales databases*

7.1.1. *Querying stores7 with DB ontologies*

To test the scalability of OntoGrate for query answering, we populated the Stores7 database (running on a local PC) with 1,000,000 synthetic data records.[c] Then we tested queries that returned result sets of varying sizes. The time that our system took to process the queries is shown in Fig. 4.

The queries we used for testing have different complexities in terms of different number of table joins and subgoals (i.e., the number of related attributes). As Fig. 4 shows, when the queries are translated to SQL, they need different numbers of table joins (i.e., from one to three) and subgoals. Some queries are only related to one table, such as a simple query one might pose to the system: "What are the names of customers in the Stores7 database living in the city of Eugene?" Suppose the query is using the Stores7 DB ontology and in OWL-QL abstract syntax:

```
premise: Customer(!C),
         (customercity !C "Eugene")
queryPattern: {(customerfname !C ?x)
               (customerlname !C ?y)}
answerKBPattern: {http://...stores7.owl}
```

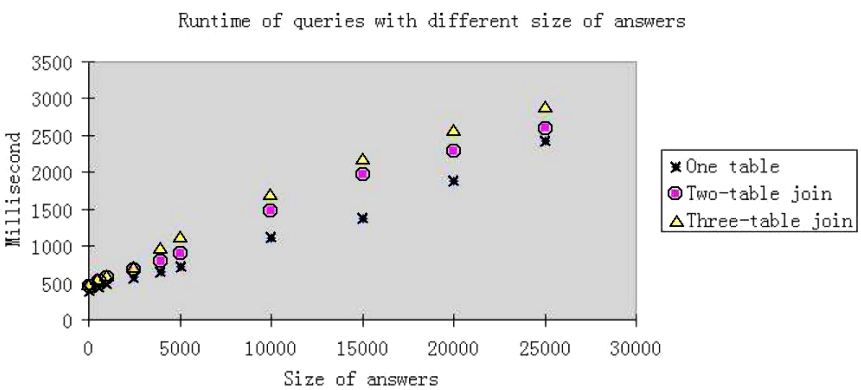

Fig. 4. The runtime of different size of answers.

[c]All experiments were performed on a 1.8 Ghz Centrino processor with 1 Gb of RAM and a MySQL database engine with a Java/JDBC implementation.

It means: given the Stores7 DB, the above query asks for the first and last names (i.e., ?x, ?y which are called must-bind variables in OWL-QL that require the answers to be returned) of customers living in Eugene. Note, the identification information (e.g., ID) of each customer is not required to be returned because !C is a *non-bind* variable in OWL-QL. The same query in SPARQL syntax would be:

```
SELECT ?x ?y
WHERE
{  ?c customerfname ?x .
   ?c customerlname ?y .
   ?c customercity "Eugene".}
```

With our syntax wrapper, it can be translated to SQL:

```
SELECT Customer.customerlname, Customer.customerfname
FROM stores7.Customer
WHERE Customer.customercity="Eugene"
```

The above query only needs the `Customer` table in Stores7 and it has 2 subgoals (i.e., `customerfname` and `customerlname`). But answering the following query ("What are the item numbers of the items sold with the quantity as 1?") needs more tables. In OWL-QL abstract syntax, the query looks like:

```
queryPattern: {
       (ordercustomernumber !Order !number)
       (customernumber !Customer !number)
       (ordernumber !Order !onumber)
       (itemordernumber !Item !onumber)
       (Itemnumber !Item ?x)
       (itemquantity !Item "1")}
answerKBPattern: {http://...stores7.owl}
```

Be translated to SQL, it is:

```
SELECT Item.itemnumber
FROM  Order, Customer, Item
WHERE Order.ordercustomernumber = Customer.customernumber and
Order.ordernumber = Item.itemordernumber and Item.itemquantity = "1"
```

Therefore it needs to join `Customer`, `Order` and `Item` tables in Stores7 and it has 5 subgoals. To get 25,000 answers from 1,000,000 records needs about 3000 milliseconds. This result is consistent with the general query answering performance of commercial databases and it should be scalable in even larger-sized data. The syntax translation of the different queries from OWL-QL to Web-PDDL using PDDOWL is less than 500 milliseconds (including loading the ontologies) and the syntax translation from Web-PDDL to SQL using PDDSQL is less than 10 milliseconds. The figure basically shows that the syntax translation roughly gives 0.5 second overhead to the whole query answering process which is acceptable to a general

commercial database engine (e.g., MySQL) and potential users (e.g., Semantic Web agents). As we expected, the running time is linear to the size of the answer.

7.1.2. *Integrating stores7 and northwind with the merged ontology*

To test the data integration function of OntoGrate for this test case, we also populated about 30,000 tuples into the Norhwind database. The data can be thought of as the subset of Stores7 and they are basically the same but in different schemas. Taking partial data and DB ontologies as input, our mapping system discovers 17 mappings. Two examples are:

(1) $\forall c, s \ @stores7{:}Customer(c) \leftrightarrow @nwind{:}Customer(c)$
(2) $\forall c, s \ @stores7{:}Customer(c) \wedge String(s) \wedge @stores7{:}customernumber(c, s) \rightarrow @nwind{:}customerid(c, s)$

Among 17 discovered rules, two are incorrect according to the gold standard (i.e., 18 rules) specified by humans. To measure the accuracy of the methods, we used typical measures *precision* and *recall*. Given source and target schemas, a mapping system will generate a mappings set P and domain experts will specify the gold standard mappings set R. The two measures are computed as: precision $= \frac{|P \bigcap R|}{|P|}$ and recall $= \frac{|P \bigcap R|}{|R|}$. The precision of the result is 88% and recall is 83%.

Based on these mappings we generated a merged ontology Stores7-Northwind with the Stores7 and Northwind ontologies. The mappings between the merged ontology and Stores7 or Northwind will be generated. For example, rule (2) from the generated mappings will be converted to:

(2a) $\forall c, s \ @stores7\text{-}nwind{:}Customer(c) \wedge String(s) \wedge @stores7\text{-}nwind{:}customernumber(c, s) \rightarrow @stores7{:}customernumber(c, s)$
(2b) $\forall c, s \ @stores7\text{-}nwind{:}Customer(c) \wedge String(s) \wedge @stores7\text{-}nwind{:}customernumber(c, s) \rightarrow @nwind{:}customerid(c, s)$

The queries we used for integration have different complexities. Similar to the tests reported above, when the queries are finally translated to SQL, they need different numbers of table joins and subgoals. Compared with the tests for querying databases with DB ontologies, the overhead is the inferential query translation from Stores7-Northwind to Stores7 and Northwind. However, the tests show that the inferential query translation of queries with various complexities in terms of different numbers of table joins and subgoals always takes less than 10 milliseconds. It shows that the time of query translation is not related to the size of the answers and it is fast enough with respect to the whole query answering process. Therefore the performance with respect to number of answers basically depends on the speed of the database itself.

7.2. *Two NBA databases — highly overlapping data*

In this case, we retrieved data from two online data repositories: NBA Official Site[d] and Yahoo Sports NBA Site[e] and create two databases accordingly. Note that the data acquisition itself is a non-trivial task. We extracted data from websites and fill in the databases. The two databases describe almost the same real-world entities, such as NBA teams, players and matches and their data are highly overlapping. There are about 6000 data instances in each database which describe information of 343 players, 100 games, 30 teams and 30 arenas.

To test OntoGrate in this real-world scenario, we first transformed the schemas to DB ontologies and applied our ontology mapping system to discover mappings. Our mapping system generated 38 matching groups and finally discovered 19 rules. There are two rules considered as wrong according to 17 gold standard rules. Therefore, recall is 100% and precision is 89%. Compared with 17 user-specified gold standard rules, the result is satisfactory. Some interesting rules are

(1) $\forall x, y\ @NBA{:}Team(x) \wedge String(y) \wedge @NBA{:}city(x, y) \rightarrow$
$@YAHOO{:}Team(x) \wedge String(y) \wedge @YAHOO{:}location(x, y)$
(2) $\forall x, y\ @NBA{:}Scores(x) \wedge @NBA{:}Team(y) \wedge @NBA{:}visteam(x, y) \rightarrow$
$@YAHOO{:}Scores(x) \wedge @YAHOO{:}Team(y) \wedge @YAHOO{:}team1(x, y)$
(3) $\forall x, y\ @NBA{:}Scores(x) \wedge @NBA{:}Team(y) \wedge @NBA{:}hometeam(x, y) \rightarrow$
$@YAHOO{:}Scores(x) \wedge @YAHOO{:}Team(y) \wedge @YAHOO{:}team2(x, y)$

The property names *@NBA:city* and *@YAHOO:location* in rule (1) do not have high similarity in their names but our system discovered it with the help of the incorporation of data mining techniques. Rule (2) and rule (3) are also interesting since even experts cannot quite tell whether *visteam* should be matched to *team*1 or *team*2 without going through the instances.

Then we put those mapping rules as bridging axioms into the merged NBA-YAHOO ontology and we could test the query answering with different queries. Finally we put the answers from two databases together. Since two databases share large portions of data, we needed to run the object reconciliation function in OntoGrate to remove the redundant data instances (e.g., NBA site and YAHOO site both have a Laker player called "Kobe Bryant").

7.3. *Mouse and zebrafish gene databases — partial overlapping data*

In this test case, we obtained the data from MGI [1], a research group on mice genes, and ZFIN [6], a research group on zebrafish genes. Gene domain researchers

[d] http://www.nba.com.
[e] http://sports.yahoo.com/nba.

normally gather data from several different species and analysis of the related gene expressions and phenotypes is a common task. However, different groups usually provide their own interface and database to access data. Therefore understanding how data from different data sources are related is very important for the domain experts. The number of tables of these two databases are very large: ZFIN has 140 and MGI has 191 tables, and some tables have at least 1 million data instances. However, based on the domain knowledge of human experts, only a small number of tables are potential to be mapped to each other. Therefore, we created smaller databases by selecting only the relations related to *marker* and *gene expressions* following suggestions from domain experts. We used about 400 data instances from ZFIN and 550 data instances from MGI.

To test OntoGrate in this interesting biomedical data scenario, we transformed the partial schemas to DB ontologies and applied our ontology mapping system to discover mappings. Our mapping system generated 2 rules. There is no rule considered as wrong and no missing rule compared to the rules provided by domain experts. Therefore, both recall and precision are 100%. The importance of this case study is not how many rules we could discover but whether these rules are meaningful and interesting to domain experts. For example, one interesting rule is

$$\forall x, y \ @MGI{:}MRK\_Marker(x) \wedge String(y) \wedge @MGI{:}symbol(x, y) \rightarrow \\ @ZFIN{:}marker(x) \wedge String(y) \wedge @ZFIN{:}mrk\_abbrev(x, y)$$

This rule shows that the marker symbol in the MGI database is matched to mrk_abbrev in the ZFIN database, where "markers" are generally referred to as DNA sequences including genes.

Then we put those mapping rules as bridging axioms into the merged ZFIN-MGI ontology and we could test the query answering with different queries. The returned answers can be put together by distinguishing them with namespaces. Genetic scientists want to see which database (i.e., model organism) the answers come from. They do not want a system to reconcile the gene sequences, expressions or functions for them.

8. Discussions and Future Work

In this section, we discuss some interesting observations and future work.

8.1. *Querying databases with existing OWL ontologies*

In a slightly different use-case scenario, we might envision a Semantic Web agent using an existing OWL ontology to access existing databases. For example, it was easy to find an existing Order ontology[f] written in OWL and published on the Web.

[f] http://www.dayf.de/2004/owl/order.owl.

This ontology is similar to the Stores7 DB ontology and with the proper mapping rules, it can be used by the Semantic Web agent to query the Stores7 database directly. In this scenario, the OWL-QL query described by the Order ontology needs to be translated to a query in the Stores7 DB ontology first. If the mappings between the Order ontology and the Stores7 DB ontology are not available, the ontology mapping process described in Sec. 5 is first conducted. Given the mappings, the remaining task reduces to the same inferential query translation process as described in Sec. 6.

However, the caveat in this situation is that our mapping rule discovery component requires that data instances from each source share some overlap. But the Order ontology has no data, making the mapping rule discovery process more difficult. There are approaches that do not have this limitation. For example, the Semap research by An *et al.* [8] provides methods to construct complex mapping rules between database schemas and their semantic models given an initial set of correspondences or matchings.

8.2. *Uncertainty of mappings*

Our mapping system or any automatic mapping system cannot generate complete and 100% accurate mappings. Without human involvement for mapping correction, if we directly input the generated mappings to our inference engine, there will be some uncertainty. It is a major future work in our data integration and data mining study. The general idea is that we will use both fuzzy logic and probabilistic theory to represent mappings with uncertainty. Then fuzzy and probabilistic reasoning will be used for data integration.

For example, the mappings from the ontology mapping system have some accuracy (confidence) based on the data mining process. We can treat the mapping rules as predictions with a probabilistic interpretation. For example, *customername* is related to *customerfname* and *customerlame* because there is a large percentage of data about the reconciled individuals suggesting that *name*, *firstname* and *lastname* satisfy the relationship $name = concatenation(firstname, lastname)$. One way to represent this mapping is to assign a probability value to this functional relationship (mapping), which shows the accuracy of the rule verified by data mining.

We also observe that our ontology mapping system assumes overlapping data among ontologies exists. If there is no overlapping data supplied by the databases under consideration, it will be very hard for mining algorithms to produce helpful results. For example, if we say the concept *location* from one ontology is mapped (related) to the concept *city* in another ontology, the mapping may only be considered as a subjective perception (belief) without the support of overlapping data instances. One way to represent this uncertain mapping (heterogeneity) is to use fuzzy logic [56] to indicate the degree of belief in how much they are related or mapped. We propose the fuzzy mapping to model the degree of belief in how two concepts are mapped to each other. We view the potential mappings as a

fuzzy set. A fuzzy value $f \in [0, 1]$ models the grade of membership of each single mapping.

9. Conclusions

In this paper. We describe our ontology-based information integration framework, OntoGrate, which systematically combines an ontology mapping system, an inference engine and several syntax wrappers to answer ontology-based queries using the data from heterogeneous databases in a highly automatic way. We have made several major contributions:

(1) We designed an ontology-based framework that provides a unified semantics between ontology mapping and information integration by transforming database schemas to Semantic Web ontologies.
(2) We developed a highly automatic ontology mapping system which leverages object reconciliation and multi-relational data mining.
(3) We completed the conjunctive query translation algorithm which uses inference to translate conjunctive queries between ontologies. This allowed our system to scale to large numbers of instances, especially for query answering over relational databases via DB ontologies.
(4) We performed several experiments as a proof of concept for OntoGrate on both synthetic and real-world data. The results demonstrate the usefulness of a unified framework for integrating both Semantic Web and database information.

The advantages of our framework will be investigated in large-size relational databases of several scientific domains, such as neuroscience and genetics. Besides considering the uncertainty in the integration process, we will focus on how to use our framework and OntoGrate to help domain experts answer scientific questions with ontology-based inferences, data mining and their databases.

Acknowledgments

We thank Shiwoong Kim, Haishan Liu and Daya Wimalasuriya for their contributions for OntoGrate. We thank Jeff Z. Pan at the University of Aberdeen for helpful discussion on OWL-QL and query answering. We also thank the ZFIN and MGI groups for providing us the genetic data and domain knowledge for the experiments.

References

[1] MGI: Mouse Genome Informatics, http://www.informatics.jax.org/.
[2] OWL Web Ontology Language, http://www.w3.org/TR/owl-ref/.
[3] Rule Interchange Format (RIF), http://www.w3.org/2005/rules/.
[4] SPARQL Query Language for RDF, http://www.w3.org/TR/rdf-sparql-query/.
[5] SWRL: A Semantic Web Rule Language Combining OWL and RuleML, http://www.w3.org/Submission/SWRL/.
[6] ZFIN: The Zebrafish Information Network, http://www.zfin.org.

[7] S. Abiteboul, R. Hull and V. Vianu, *Foundations of Databases* (Addison-Wesley, 1995).
[8] Y. An, A. Borgida, R. J. Miller and J. Mylopoulos, A semantic approach to discovering schema mapping expressions, in *Proceedings of the 23rd International Conference on Data Engineering*, 2007, pp. 206–215.
[9] P. Atzeni and V. DeAntonellis, *Relational Database Theory: A Comprehensive Introduction* (Bejamin Cummings, 1993).
[10] C. Beeri and M. Y. Vardi, Formal systems for tuple and equality generating dependencies, *SIAM Journal on Computing* **13**(1) (1984) 76–98.
[11] T. Berners-Lee, J. Hendler and O. Lassila, The Semantic Web, *Scientific American* **284**(5) (2001).
[12] P. A. Bernstein and E. Rahm, Data warehouse scenarios for model management, in *Proceedings of International Conference on Conceptual Modeling*, 2000, pp. 1–15.
[13] P. Bouquet, F. Giunchiglia, F. van Harmelen, L. Serafini and H. Stuckenschmidt, Contextualizing ontologies, *J. Web Sem.* **1**(4) (2004) 325–343.
[14] J. D. Bruijn and A. Polleres, Towards an ontology mapping specification language for the semantic web, Technical report, Digital Enterprise Research Institute, June 2004.
[15] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi and R. Rosati, Knowledge representation approach to information integration, in *Proceedings of AAAI Workshop on AI and Information Integration*, 1998, pp. 58–65.
[16] R. Dhamankar, Y. Lee, A. Doan, A. Y. Halevy and P. Domingos, iMAP: Discovering complex mappings between database schemas. in *Proceedings of the ACM Conference on Management of Data*, 2004, pp. 383–394.
[17] A. Doan, P. Domingos and A. Y. Halevy, Reconciling schemas of disparate data sources: A machine-learning approach, in *Proceedings of the ACM Conference on Management of Data*, 2001, pp. 509–520.
[18] A. Doan, J. Madhavan, P. Domingos and A. Y. Halevy, Learning to map between ontologies on the semantic web, in *Proceedings of the International World Wide Web Conferences*, 2002, pp. 662–673.
[19] X. Dong, A. Y. Halevy and J. Madhavan, Reference reconciliation in complex information spaces, in *Proceedings of the ACM Conference on Management of Data*, 2005, pp. 85–96.
[20] D. Dou and P. LePendu, Ontology-based integration for relational databases, in *Proceedings of the 2006 ACM Symposium on Applied Computing*, 2006, pp. 461–466.
[21] D. Dou, P. LePendu, S. Kim and P. Qi, Integrating databases into the semantic web through an ontology-based framework, in *Proceedings of the Third International Workshop on Semantic Web and Databases (SWDB'06)*, pp. 54, 2006, co-located with ICDE 2006.
[22] D. Dou, D. V. McDermott and P. Qi, Ontology translation on the semantic web, *Journal on Data Semantics* **2** (2005) 35–57.
[23] E. Dragut and R. Lawrence, Composing mappings between schemas using a reference ontology, in *Proceedings of International Conference on Ontologies, Databases and Application of Semantics (ODBASE)*, 2004, pp. 783–800.
[24] J. Euzenat and P. Shvaiko, *Ontology Matching* (Springer-Verlag New York, 2007).
[25] R. Fagin, P. G. Kolaitis, R. J. Miller and L. Popa, Data Exchange: Semantics and Query Answering, in *Proceedings of International Conference on Database Theory*, 2003, pp. 207–224.
[26] C. Fellbaum, editor, *WordNet: An Electronic Lexical Database* (MIT Press, 1998).

[27] R. Fikes, P. J. Hayes and I. Horrocks, OWL-QL — a language for deductive query answering on the semantic web, *J. Web Sem.* **2**(1) (2004) 19–29.
[28] L. M. Haas, Beauty and the beast: The theory and practice of information integration, in *Proceedings of the International Conference of Database Theory*, 2007, pp. 28–43.
[29] L. M. Haas, M. A. Hernandez, H. Ho, L. Popa and M. Roth, Clio grows up: From research prototype to industrial tool, in *Proceedings of the ACM Conference on Management of Data*, 2005, pp. 805–810.
[30] A. Y. Halevy, Z. G. Ives, D. Suciu and I. Tatarinov, Schema mediation in peer data management systems, in *Proceedings of International Conference on Data Engineering*, 2003, pp. 505–516.
[31] W. Hu and Y. Qu, Block matching for ontologies, in *Proceedings of the International Semantic Web Conference*, 2006, pp. 300–313.
[32] P. G. Kolaitis, Schema mappings, data exchange, and metadata management, in *Proceedings of Principles of Database Systems*, 2005, pp. 61–75.
[33] M. Lenzerini, Data Integration: A Theoretical Perspective, in *Proceedings of Symposium on Principles of Database Systems*, 2002, pp. 233–246.
[34] L. Lubyte and S. Tessaris, Extracting ontologies from relational databases, in *Proceedings of Description Logics*, 2007, pp. 122–126.
[35] J. Madhavan, P. A. Bernstein and E. Rahm, Generic schema matching with cupid, in *Proceedings of Very Large Data Bases Conference*, 2001, pp. 49–58.
[36] A. Maedche, B. Motik, N. Silva and R. Volz, MAFRA — A mapping framework for distributed ontologies, in *Proceedings of International Conference on Knowledge Engineering and Knowledge Management*, 2002, pp. 235–250.
[37] D. V. McDermott and D. Dou, Representing disjunction and quantifiers in RDF, in *Proceedings of International Semantic Web Conference*, 2002, pp. 250–263.
[38] D. L. McGuinness, R. Fikes, J. Rice and S. Wilder, The chimaera ontology environment, in *Proceedings of the National Conference on Artificial Intelligence*, 2000, pp. 1123–1124.
[39] S. A. McIlraith and D. L. Martin, Bringing semantics to web services, *IEEE Intelligent Systems* **18**(1) (2003) 90–93.
[40] R. J. Miller, M. A. Hernández, L. M. Haas, L.-L. Yan, C. T. H. Ho, R. Fagin and L. Popa, The CLIO project: Managing heterogeneity, *SIGMOD Record* **30**(1) (2001) 78–83.
[41] B. Motik, I. Horrocks and U. Sattler, Bridging the gap between owl and relational databases, in *Proceedings of the International World Wide Web Conference*, 2007, pp. 807–816.
[42] S. Nijssen and J. N. Kok, Efficient frequent query discovery in farmer, in *Proceedings of European Conference on Principles and Practice of Knowledge Discovery in Databases*, 2003, pp. 350–362.
[43] N. F. Noy and M. A. Musen, PROMPT: Algorithm and tool for automated ontology merging and alignment, in *Proceedings of the National Conference on Artificial Intelligence*, 2000, pp. 450–455.
[44] G. Pierre, Context representation in domain ontologies and its use for semantic integration of data, *Journal Data Semantics* **10** (2008) 174–211.
[45] A. Poggi, D. Lembo, D. Calvanese, G. D. Giacomo, M. Lenzerini and R. Rosati, Linking data to ontologies, *Journal Data Semantics* **10** (2008) 133–173.
[46] R. Pottinger and A. Levy, A scalable algorithm for answering queries using views, in *Proceedings of the Very Large Data Bases Conference*, 2000, pp. 484–495.
[47] W. J. Premerlani and M. R. Blaha, An approach for reverse engineering of relational databases, *Commun. ACM Journal* **37**(5) (1994) 42–49, 134.

[48] H. Qin, D. Dou and P. LePendu, Discovering executable semantic mappings between ontologies, in *Proceedings of the International Conference on Ontologies, Databases and Application of Semantics (ODBASE)*, 2007, pp. 832–849.

[49] E. Rahm and P. A. Bernstein, A survey of approaches to automatic schema matching, *Very Large Data Bases Journal* **10**(4) (2001) 334–350.

[50] R. Reiter, Towards a logical reconstruction of relational database theory, in M. L. Brodie, J. Mylopoulos and J. W. Schmidt, (eds.), *On Conceptual Modelling, Perspectives from Artificial Intelligence, Databases, and Programming Languages*, Topics in Information Systems, Springer, 1984, pp. 191–233.

[51] S. Russell and P. Norvig, *Artificial Intelligence: A Modern Approach* (Prentice-Hall, 1995).

[52] A. P. Sheth and J. A. Larson, Federated database systems for managing distributed, heterogeneous, and autonomous databases, *ACM Computing Surveys* **22**(3) (1990) 183–236.

[53] G. Stoilos, G. B. Stamou and S. D. Kollias, A string metric for ontology alignment, in *Proceedings of the International Semantic Web Conference*, 2005, pp. 624–637.

[54] L. Stojanovic, N. Stojanovic and R. Volz, Migrating data-intensive web sites into the semantic web, in *Proceedings of ACM Symposium on Applied Computing* (ACM Press, 2002), pp. 1100–1107.

[55] H. Stuckenschmidt and M. Uschold, Representation of semantic mappings, in *Proceedings of the Semantic Interoperability and Integration*, 2005, pp. 111–117.

[56] L. A. Zadeh, Fuzzy logic, *Computer* **21**(4) (1988) 83–93.