Ontology-based Integration for Relational Databases

Dejing Dou
Computer and Information Science
University of Oregon
Eugene, Oregon 97403, USA
dou@cs.uoregon.edu

Paea LePendu
Computer and Information Science
University of Oregon
Eugene, Oregon 97403, USA
paea@cs.uoregon.edu

ABSTRACT

In this paper, we show that representation and reasoning techniques used in traditional knowledge engineering and the emerging Semantic Web can play an important role for heterogeneous database integration. Our OntoGrate architecture combines ontology-based schema representation, first order logic inference, and some SQL wrappers to integrate two sample relational databases. We define *inferential data integration* as the theoretical framework for our approach. The performance evaluation for query answering shows that OntoGrate reformulates conjunctive queries and retrieves over 100,000 answers from a target database in under 30 seconds. In addition to query answering, the system translates 40,000 database facts from source to target in under 30 seconds.

Categories and Subject Descriptors

H.2.5 [**Database Management**]: Heterogeneous Databases; H.3.5 [**Information Storage and Retrieval**]: Online Information Services—*data sharing*; I.2.3 [**Artificial Intelligence**]: Deduction and Theorem Proving—*inference engines*; I.2.4 [**Artificial Intelligence**]: Knowledge Representation Formalisms and Methods—*representation languages, predicate logic*

General Terms

Data Integration, Ontology

Keywords

semantic integration, first order logic, inference

1. INTRODUCTION

Autonomous database systems usually have incompatible schemas making interoperability among them difficult. This has long been recognized as a schema mapping and data integration problem [7, 14]. In simplest terms, database integration requires (i) mapping systems that define the relationships (mappings) among database schemas and (ii) integration systems that use those mappings to answer queries or translate data across database sources. To make things more interesting, in the foreseeable future, databases, knowledge bases, the World Wide Web and the Semantic Web [1] will coexist and agents may want to integrate these.

Mapping systems help to construct a global view of data by establishing inter-schema correspondences and mappings [20]. Popular methods represent mappings as SQL views [2, 11, 6], which can be directly invoked by database management systems. Although this gives us the satisfaction of good performance, especially with query rewriting algorithms such as [19], these techniques are limited to only database information systems.

Others approach data integration using ontologies or logic-based frameworks such as description logic or Datalog [3, 15]. In addition to the more expressive representations offered by ontologies, formal logics provide inference mechanisms for reasoning on and preservation of the semantics being expressed. This allows integration to cover a larger variety of structured data in theory, although it raises the question of adequate performance in real systems. With the possible exception of CARNOT [5], there are no notable implementations to our knowledge which use this approach for database systems.

We adopt an approach similar to one used in ontology translation for the Semantic Web [9]. To model database schemas, concepts, and the relationships (mappings) among them, we use the Web-PDDL ontology language. This is a first order logic language we use for the OntoEngine inference engine. As far as we know, ours is the only approach that successfully uses first order logic and inference for database integration, and we call it *inferential data integration*.

To make our approach practical, we developed some rudimentary SQL wrappers that allow our system to access real database systems. Our preliminary experiments of the OntoGrate prototype demonstrates that such an integration approach is actually tractable for real systems.

2. ONTOLOGIES

An ontology is the formal specification of the vocabularies of concepts and the relationships among them. Ontologies have played a key role for describing semantics of data in both traditional knowledge engineering and the emerging Semantic Web. In this section, we describe the ontology language Web-PDDL and show how this language can represent not only database schemas but also semantic mappings between them.

To illustrate our approach, we use examples of two heterogeneous relational database schemas: Stores7[1] from Informix and Nwind[2] (Northwind) from Microsoft. Both of these are from the sales application domain, such as an online shopping cart system

[1] http://www.cs.uoregon.edu/~paea/research/stores7.png
[2] http://www.cs.uoregon.edu/~paea/research/nwind.png

dealing with customers, orders and products. Figure 1 shows an intuitive diagram of how these schemas and their relationships can be represented as ontologies and mapping rules.

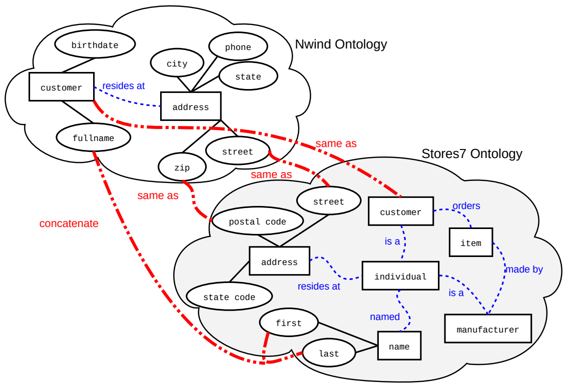


Figure 1: The customer portions of two sales ontologies and the mappings between them. Each cloud represents an ontology. Dotted or dashed lines represent relationships between concepts. There are relationships within each ontology, as well as ones between the two ontologies. The inter-ontology relationships are referred to more specifically as mapping rules.

2.1 Web-PDDL

Originally, Web-PDDL [18] was designed as a language for representing Semantic Web ontologies, their mappings, data instances and queries. It is a strongly typed first order logic language with Lisp-like syntax. It extends the Planning Domain Definition Language of [17] with XML namespaces, multi-type inheritance and more flexible notations for axioms. Though designed with Semantic Web documents in mind, Web-PDDL also can be used to represent a database schema, such as Stores7, as an ontology shown in Figure 2[3]. For example, the axiom listed describes the foreign key constraint between the State and Customer relations in Stores7.

```
(define (domain stores7-ont)
  (:extends
    (uri "http://www.cs.uoregon.edu/~paea/research/sql.pddl#"
      :prefix sql) ...)
  (:types
    Customer State - @sql:Relation
    String - @sql:Varchar ...)
  (:predicates
    (customerfname x - Customer y - @sql:varchar)
    (customerlname x - Customer y - @sql:varchar)
    (customerstatecode x - Customer y - @sql:varchar) ...)
  (:axioms
    (forall (c - Customer code - @sql:varchar)
      (if (customerstatecode c code)
        (exists (s - State) (statecode s code)))) ...)
  (:facts
    (@sql:primarykey Customer "customernumber") ...))
```

Figure 2: The Stores7 schema as an ontology.

A *merged ontology* [9] is the ontology equivalent of a global view over local schemas. It consists of common elements from a source ontology and a target ontology but also defines the semantic mappings between them as *bridging axioms* (sometimes called "articulation axioms"). A merged ontology allows all the relevant symbols in a domain to interact so that facts can be translated from one ontology to another using inference over the bridging axioms[4].

[3]Discussed in more detail in Section 2.2.1.
[4]See Section 3 for more details on inference.

Suppose we have the mappings between Nwind and Stores7 given in Figure 3.

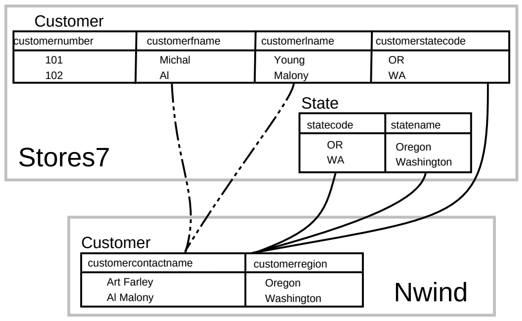


Figure 3: Mappings between Stores7 and Nwind. Although not technically an ontology mapping representation, this diagram roughly illustrates the relationships between inter-schema names and address with actual data. We refer to this example throughout the paper.

We can write bridging axioms in Web-PDDL to express these relationships. For example, to represent the 3-way mapping from Nwind's *customerregion* to Stores7's *statename*, *statecode* and *customerstatecode*, we can use the Web-PDDL expression in Figure 4, where @nwind and @stores7 are namespace prefixes for each ontology[5]. We can put all bridging axioms into the new, merged ontology called Stores7-Nwind.

```
(forall (x - @nwind:Customer y - @sql:varchar)
   (if (@nwind:customerregion x y)
      (exists (z - @stores7:State t - @sql:varchar)
         (and (@stores7:customerstatecode x t)
            (@stores7:statename z y)
            (@stores7:statecode z t)))))
```

Figure 4: Example of a bridging axiom.

This mapping is quite complex in that it uses the semantics of the Stores7 ontology to lookup, join and resolve the *statecode* and *statename* bindings based on a *customerregion* from the Nwind ontology. In other words, even though there is no such concept of state code abbreviations in Nwind, we can still resolve this information.

2.2 Schemas, Ontologies and Mappings

So far we have described the Web-PDDL language for defining ontologies and bridging axioms. In this section, we relate these ideas more closely to how to represent database schemas as ontologies and how to define schema mappings as bridging axioms.

2.2.1 Schemas as Ontologies

A database schema can be represented as an ontology using some very simple transformation rules shown in Figure 5. The Stores7 (as mentioned in Figure 2) and Nwind schemas were transformed to ontologies[6] entirely by using just these rules. Although these simple rules appear to work well for our sample databases, we still believe that human assistance may be required for some complex semantic interpretations.

[5]We assume that a type equivalence axiom (not shown) also exists between the two Customer relations.
[6]http://www.cs.uoregon.edu/~paea/research/stores7.pddl
http://www.cs.uoregon.edu/~paea/research/nwind.pddl

$$\begin{aligned} \text{Relation} &\leftrightarrow \text{Type} && (1)\\ \text{Attribute} &\leftrightarrow \text{Predicate} && (2)\\ \text{Integrity Constraint} &\leftrightarrow \text{Axiom} && (3)\\ \text{Primary Key} &\leftrightarrow \text{Fact} && (4) \end{aligned}$$

Figure 5: Schema to Ontology translation rules.

Inheritance was briefly mentioned in Section 2.1. Database ontologies like the ones we defined can inherit concepts from a more general SQL[7] ontology which defines basic notions such as relations, data types (integer, varchar, decimal, etc.), keys and aggregate functions.

2.2.2 Mappings as Bridging Axioms

Unlike the process of transforming schemas to ontologies, defining semantic relationships between concepts is much too subtle for full automation. Some tools do exist (see Section 6) to facilitate the process, but human interaction is invariably required. For this study, discovering mapping rules is not our focus, so we have manually defined the bridging axioms based on our understanding of the schemas.

Figure 6 shows some example bridging axioms between Stores7 and Nwind written in Web-PDDL. The first "T->" axiom means that Stores7 and Nwind have equivalent types. The other axioms essentially describe all mappings depicted in Figure 3. Notice that while we claim the `Customer` types are equivalent, the bridging axioms also tell us how their attributes differ. The *customercity* axiom is a 1-1 mapping, whereas the *customercontactname* one is a 2-1 mapping which concatenates first and last names. The last axiom is one we have previously discussed in Section 2.1.

```
(T-> @stores7:Customer @Nwind:Customer)

(forall (c - @stores7:Customer city - @sql:varchar)
   (iff (@stores7:customercity c city)
      (@nwind:customercity c city)))

(forall (c - @stores7:Customer f l - @sql:varchar)
   (if (and
         (@stores7:customerfname c f)
         (@stores7:customerlname c l))
      (@nwind:customercontactname c (@sql:concat f l))))

(forall (c - @nwind:Customer region - @sql:varchar)
   (if (@nwind:customerregion c region)
      (exists (s - @stores7:State code - @sql:varchar)
      (and
         (@stores7:customerstatecode c code)
         (@stores7:statename s region)
         (@stores7:statecode s code)))))
```

Figure 6: Bridging axioms for Figure 3.

Although we often refer to using a global view in this paper, bridging axioms are flexible enough to use in peer-to-peer data management scenarios [12] by mapping ontologies directly to each other without having to define a global view over them.

3. INFERENTIAL DATA INTEGRATION

Given the merged ontology between two schemas expressed in the first order ontology language, Web-PDDL, we can now perform integration using first order theory. In this section we specify the problem of integration and show how inference can solve it, forming the basis of our *inferential data integration* framework.

3.1 Information Integration

Information integration is regarded as combining data residing at different sources and providing a unified access to these data through a reconciled global view [13, 14]. For relational databases, this can be decomposed into two sub-problems:

Query Translation: The process of extracting data expressed using one schema to answer the query in another schema.

Data Translation: Translating data from a source schema to a target schema for the purpose of information exchange.

The problem of query translation has been the focus of most recent work, but data translation is also important [7]. We stipulate that both of these problems can be addressed using inference.

3.2 Inference

Suppose we have used bridging axioms to describe the mappings between two schemas, Schema S from DB_s and Schema T from DB_t (such as those in Figure 6 for Stores7 and Nwind). Let the set of bridging axioms be denoted M_{s_t}. Let the symbol $\leadsto_D$ indicate data translation between two schemas (such as S and T) so that: $\alpha_s \leadsto_D \beta_t$ means β_t is the translation of α_s, where α_s and β_t are assertions (facts) corresponding to the data instances in DB_s and DB_t. For example, the following is an assertion from Stores7 and its equivalent expressed in Web-PDDL:

The statecode of a Customer identified as 101 is OR.
customerstatecode(Customer#101, "OR")

For a set of assertions (or "dataset"), we stipulate that the translation of α_s is simply the largest set of assertions, β_t, entailed by α_s through the mapping rules M_{s_t}. A consequence of this stipulation is that

$$(M_{s\_t};\alpha_s) \leadsto_D \beta_t \text{ only if } (M_{s\_t};\alpha_s) \vDash^{8} \beta_t$$

To guarantee this requires sound inference. In other words, "$\leadsto_D$" entails soundness, so we can use $\vdash$[9] to represent data translation with our algorithm:

$$(M_{s\_t};\alpha_s) \leadsto_D \beta_t \Leftrightarrow (M_{s\_t};\alpha_s) \vdash \beta_t \Rightarrow (M_{s\_t};\alpha_s) \vDash \beta_t$$

This definition means that β_t can be derived from the mappings M_{s_t} and assertions α_s by inference.

Our proposal may seem counterintuitive, in that one might assume translating a dataset means finding an *equivalent* dataset in a different vocabulary. As justification, we point out that our standard has been taken for granted in the case of another main problem in data integration: query translation, the process of extracting data expressed using one schema to answer the query in another schema. We will use the symbol $\leadsto_Q$ to indicate the query translation. If α_s is a query in Schema S, its translation is a query β_t in Schema T such that any answer (set of bindings) to β_t is also an answer to α_s. In other words:

$$(M_{s\_t};\alpha_s) \leadsto_Q \beta_t \text{ only if } (M_{s\_t};\theta(\beta_t)) \vDash \theta(\alpha_s)$$

for any substitution θ, where $\theta(\beta_t)$ is from the target database DB_t. It also means, for any substitution θ,

$$\begin{aligned}(M_{s\_t};\alpha_s) \leadsto_Q \beta_t &\Leftrightarrow (M_{s\_t};\theta(\beta_t)) \vdash \theta(\alpha_s)\\ &\Rightarrow (M_{s\_t};\theta(\beta_t)) \vDash \theta(\alpha_s)\end{aligned}$$

We claim that β_t is the translation of α_s if and only if, for every substitution θ, $\theta(\beta_t)$ is the *weakest* statement in Schema T such that $\theta(\beta_t)$ is from DB_t, $(M_{s\_t};\theta(\beta_t)) \vdash \theta(\alpha_s)$ and $M_{s\_t} \nvdash \theta(\alpha_s)$.

[7] http://www.cs.uoregon.edu/∼paea/research/sql.pddl

[8] The logic symbol $\vDash$ can be read as "entails."

[9] The logic symbol $\vdash$ can be read as "infers."

The weakest statement means that β_t need not be (and seldom is) *equivalent* to α_s, in the sense that any answer to one is an answer to the other. All we need is that any answer to β_t be an answer to α_s. In the literature this is also known as *query containment* [14].

Since both data translation and query translation can be defined as an inference, we can call our data integration approach *inferential data integration*.

4. THE ONTOGRATE ARCHITECTURE

Shown in Figure 7, the OntoGrate architecture for database integration[10] includes the OntoEngine reasoner, and some SQL wrappers that work with our merged Stores7-Nwind ontology. Although not the focus of this paper, an interface allows the user to submit queries or data translation requests.

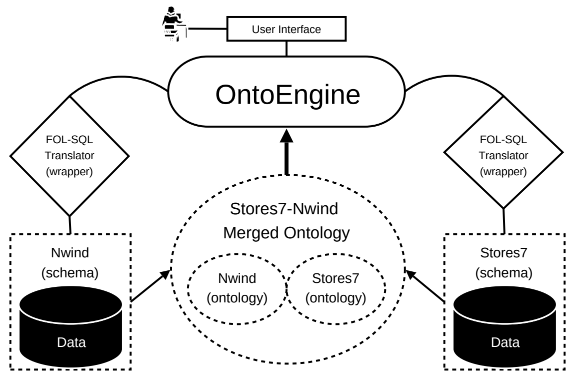


Figure 7: The OntoGrate Architecture for Database Integration

4.1 SQL Wrapper

The SQL wrapper translates between first order logic (FOL) and SQL syntax in two directions. First, it translates FOL queries to SQL queries which it then executes on the appropriate SQL database using JDBC. Second, it translates the resulting SQL record sets to FOL facts (also known as assertions). The process is purely syntactic and straightforward based on the transformation rules outlined in Figure 5.

4.1.1 Web-PDDL Query to SQL Query

Translating a Web-PDDL (FOL) query to SQL is a direct application of the transformation rules. The only unusual aspect of our translation is the automatic addition of primary keys as shown in Figure 8.

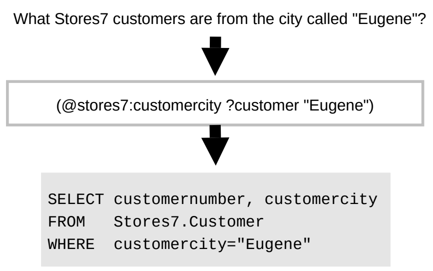
Once translated from Web-PDDL, the resulting SQL query can then be executed on the appropriate database[11] using standard JDBC APIs to obtain a result set as depicted in Figure 9. In order to be processed by OntoEngine, these tuples must be translated to Web-PDDL facts.

4.1.2 SQL Record Set to Web-PDDL Facts

Translating SQL records back into Web-PDDL facts is also a straightforward task using the transformation rules in Figure 5. The resulting facts for the *customercity* query are shown in Figure 9.

[10]Our recent work is extending OntoGrate to integrate both databases and Semantic Web resources.

[11]We assume that the appropriate database connections are identified using URIs supplied in the user environment.

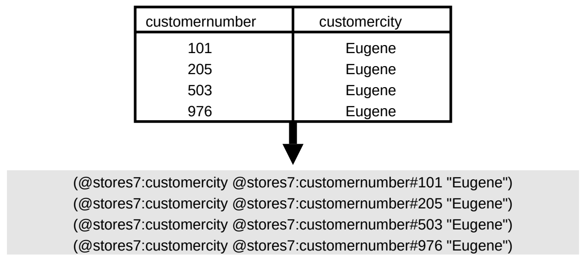


What Stores7 customers are from the city called "Eugene"?

```
(@stores7:customercity ?customer "Eugene")
```

```
SELECT customernumber, customercity
FROM   Stores7.Customer
WHERE  customercity="Eugene"
```

Figure 8: Three versions of the same query. Primary key information can be easily looked up and added to simplify object identification and join operations.



customernumber	customercity
101	Eugene
205	Eugene
503	Eugene
976	Eugene

```
(@stores7:customercity @stores7:customernumber#101 "Eugene")
(@stores7:customercity @stores7:customernumber#205 "Eugene")
(@stores7:customercity @stores7:customernumber#503 "Eugene")
(@stores7:customercity @stores7:customernumber#976 "Eugene")
```

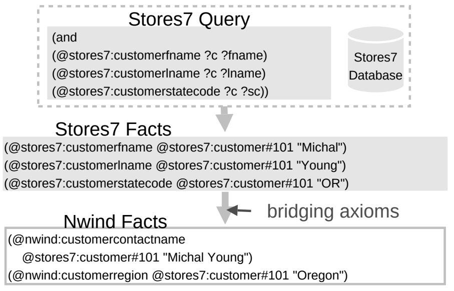
Figure 9: A SQL result set for the *customercity* query can easily be translated into Web-PDDL facts.

SQL wrappers therefore give OntoGrate direct access to databases to run queries and get facts as necessary, but to address the problem of integration, we need the OntoEngine inference engine.

4.2 OntoEngine

OntoEngine is a sound, first order theorem prover designed originally for ontology translation on the Semantic Web [9]. It implements both forward chaining and backward chaining reasoners using generalized modus ponens [21]. The reasoners help OntoGrate process skolem functions, answer queries and translate facts.

Consider the scenario where a user wants to translate customer data from Stores7 into Nwind, a typical situation when companies migrate from legacy information systems. Given a set of Stores7 facts (which might be obtained from a query), OntoGrate uses the forward chaining reasoner and bridging axioms to translate that set of source facts into the target Nwind schema. The main idea is illustrated in Figure 10.



```
(and
(@stores7:customerfname ?c ?fname)
(@stores7:customerlname ?c ?lname)
(@stores7:customerstatecode ?c ?sc))
```

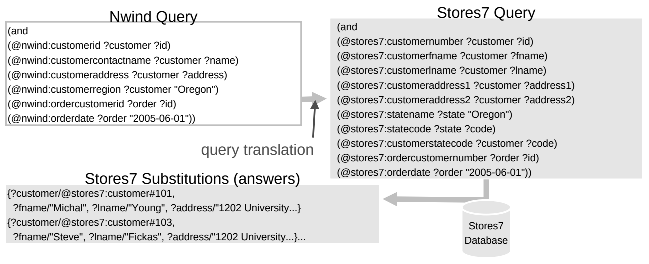
```
(@stores7:customerfname @stores7:customer#101 "Michal")
(@stores7:customerlname @stores7:customer#101 "Young")
(@stores7:customerstatecode @stores7:customer#101 "OR")
```

```
(@nwind:customercontactname
  @stores7:customer#101 "Michal Young")
(@nwind:customerregion @stores7:customer#101 "Oregon")
```

Figure 10: Facts from Stores7 are translated into Nwind's representation by using the forward chaining reasoner with bridging axioms.

On the other hand, suppose a user wants to answer a source

Nwind query using the target Store7 database, as in a peer-to-peer data management environment. In this context, OntoGrate invokes the backward chaining reasoner to reformulate the source query to the target one, then retrieves target data using the SQL wrappers to finally answer the source query. The process is shown in Figure 11.



```
(and
(@nwind:customerid ?customer ?id)
(@nwind:customercontactname ?customer ?name)
(@nwind:customeraddress ?customer ?address)
(@nwind:customerregion ?customer "Oregon")
(@nwind:ordercustomerid ?order ?id)
(@nwind:orderdate ?order "2005-06-01"))
```

```
(and
(@stores7:customernumber ?customer ?id)
(@stores7:customerfname ?customer ?fname)
(@stores7:customerlname ?customer ?lname)
(@stores7:customeraddress1 ?customer ?address1)
(@stores7:customeraddress2 ?customer ?address2)
(@stores7:statename ?state "Oregon")
(@stores7:statecode ?state ?code)
(@stores7:customerstatecode ?customer ?code)
(@stores7:ordercustomernumber ?order ?id)
(@stores7:orderdate ?order "2005-06-01"))
```

```
{?customer/@stores7:customer#101,
  ?fname/"Michal", ?lname/"Young", ?address/"1202 University...}
{?customer/@stores7:customer#103,
  ?fname/"Steve", ?lname/"Fickas", ?address/"1202 University...}...
```

Figure 11: A query asked in Nwind's representation is reformulated for Stores7, then translated into SQL for data retrieval.

5. EMPIRICAL EVALUATION

As we have shown, given a merged ontology, OntoGrate can perform database integration using custom SQL wrappers and an inference engine. We have performed some preliminary tests to verify that our approach is tractable for reasonably sized input.[12]

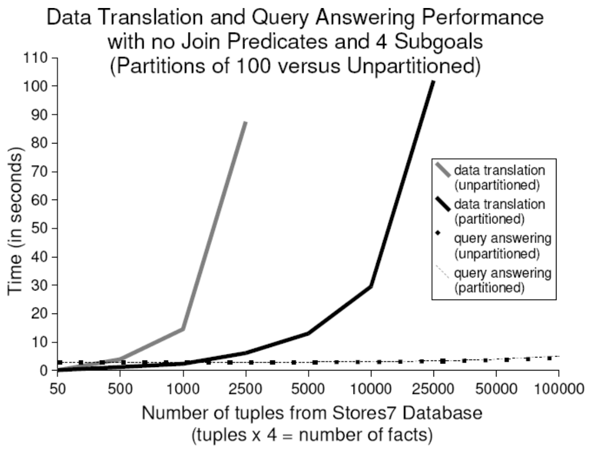


Figure 12: This is the average performance for data translation and query answering for a simple query with only 4 subgoals. Data partitioning is discussed in Section 5.3.

5.1 Methods

The Stores7-Nwind merged ontology[13] contains approximately: 13 types, 22 predicates, 25 primary key facts, and 61 bridging axioms which describe type equivalence, type subsumption, predicate equivalence and other arbitrary relationships.

We measure the performance of two types of tests: data translation and query answering. Forward chaining (data translation) is a more expensive operation so we expect slower performance.

For data translation, we use a simple query with no join predicates to retrieve facts from Stores7. Join predicates do not affect data translation performance. On the other hand, joins do significantly affect the performance of query answering, so we test two varieties of complex queries, star and chain queries. A star query contains a distinct subgoal (relation) that joins with every other subgoal in a logical star-like formation. Whereas a chain query contains subgoals that join to ones before or after it in a logical chain-like formation.

[12]All experiments were performed on a 1.8Ghz Centrino processor with 1Gb of RAM and a MySQL database engine.

[13]`http://www.cs.uoregon.edu/~paea/research/stores7_nwind_merging.pddl`

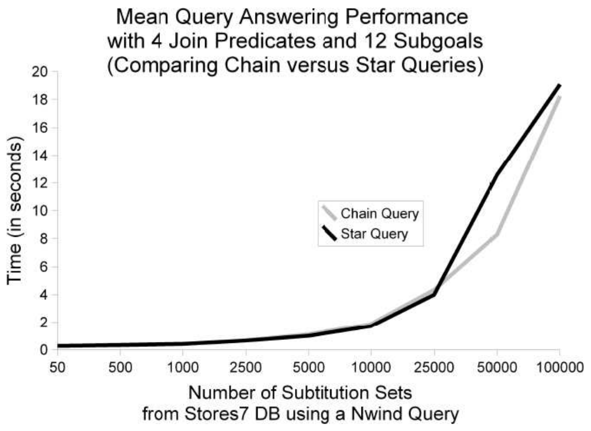


Figure 13: This figure compares query answering performance for chain and star queries using a more complex query with several joins and subgoals.

5.2 Metrics and Results

We measure performance for data translation based on the number of facts in the source being translated. Intuitively speaking, a fact corresponds roughly to a single cell in a table.

For query answering, we measure performance based on the number of substitutions retrieved to answer a query. A substitution corresponds roughly to a row in a table. Therefore, a query with 4 subgoals and 100,000 substitutions retrieved corresponds to roughly 400,000 facts (4 columns by 100,000 rows = 400,000 cells).

5.3 Discussion

Query answering (Figures 12 and 13) is remarkably fast compared to data translation. Recall that 100,000 substitutions for 12 subgoals, such as the Stores7 query in Figure 11, corresponds to over 1.2 million facts.

However, during initial tests, data translation performance was slower than observed in ontology translation for the Semantic Web. Further investigation revealed that the internal data structures used in the forward chaining algorithm were optimized for data in the Semantic Web which tends to be partitioned more randomly over the domain space. For databases, data described as, *"All customers from Eugene,"* represents a set of similar (unpartitioned) data; in particular, all the facts are bound to the object "Eugene." A more partitioned set would be like, *"All customers with zip code starting with 972,"*. In essence, the forward chaining algorithm uses data structures that are dependent on the object bindings being well distributed (something we can address in immediate future work).

We tested our hypothesis with a set of facts that are partitioned evenly in groups of 100. For example, only select 100 customers from each city. The results were considerably faster as shown by the darker solid line in Figure 12. On the other hand, the overlapping dotted lines in the figure prove that backward chaining (query answering) is unaffected by this data distribution problem.

6. RELATED WORK

Related approaches to *inferential data integration* can be found in several disciplines, listed below.

Schema mapping. Although the focus of our paper is not in finding mappings, automatic or semi-automatic schema (ontology) matching tools [20, 16, 8, 6] can be helpful in providing suggestions to a domain expert. Clio [11] is a semi-automatic tool that uses an abstract query graph representation for mappings. The COMA [10] framework combines schemas with a reference ontology and uses composition to build mappings. Our approach is to define schema mappings as first order logic axioms.

Database reverse engineering. There are very few approaches investigating the transformation of relational schemas to ontologies. One similar approach to ours is when a relational model is mapped to frame logic which can then be represented in RDF [23]. Our approaches share the same process of semantic annotation. However, our focus is not just on semantic representation for schemas or data instances, but more importantly on the integration of different relational databases.

Data integration and logic inference. General integration models such as federated databases [22], data warehouses [2] and peer-to-peer data management [12] exist. The integration process is almost always implemented with a data mediator and an integrated schema represented as view definitions. 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 brilliantly summarized and compared in [4]. While very similiar to ours, these approaches tend to be more constricted, depending on fixed global ontologies such as CYC or LOOMS or a less expressive logic such as Description Logic or Datalog.

7. CONCLUSIONS AND FUTURE WORK

We have developed an ontology-based, first order logic approach to integrate heterogeneous relational databases, which we call *inferential data integration*. We implemented the OntoGrate system to evaluate our approach by accessing real databases with large amounts of data.

Our results demonstrate that OntoGrate is promising for integrating real databases, with adequate performance for query answering. However, more work needs to be done for improving data translation performance. In particular, optimized data structures need to be developed in the forward chaining algorithm.

An interesting future direction might be to test OntoGrate against biomedical databases, where domain experts (e.g., biologists) need a straightforward way to ask more interesting questions that require several data sources to answer. On the other hand, to span the scope of information integration, we are extending OntoGrate to integrate databases, XML files and Semantic Web resources.

8. REFERENCES

[1] T. Berners-Lee, J. Hendler, and O. Lassila. The Semantic Web. *Scientific American*, 284(5):34–43, 2001.

[2] P. A. Bernstein and E. Rahm. Data warehouse scenarios for model management. In *ER 2000*, pages 1–15, 2000.

[3] D. Calvanese and G. De Giacomo. Data integration: A logic-based perspective. *AI Magazine*, 26(1):59–70, 2005.

[4] D. Calvanese, G. De Giacomo, M. Lenzerini, D. Nardi, and R. Rosati. Knowledge representation approach to information integration. In *Proc. of AAAI Workshop on AI and Information Integration*, pages 58–65. AAAI Press/The MIT Press, 1998.

[5] C. Collet, M. N. Huhns, and W.-M. Shen. Resource integration using a large knowledge base in carnot. *IEEE Computer*, 24(12):55–62, 1991.

[6] R. Dhamankar, Y. Lee, A. Doan, A. Y. Halevy, and P. Domingos. imap: Discovering complex mappings between database schemas. In *Proceedings of SIGMOD Conference 2004*, pages 383–394, 2004.

[7] A. Doan and A. Y. Halevy. Semantic-integration research in the database community: a brief survey. *AI Magazine*, 26(1):83–94, 2005.

[8] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map between ontologies on the semantic web. In *Proceedings of the World-Wide Web Conference*, 2002.

[9] D. Dou, D. V. McDermott, and P. Qi. Ontology Translation on the Semantic Web. *Journal of Data Semantics*, 2:35–57, 2005.

[10] 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.

[11] 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 2005 ACM SIGMOD International Conference on Management of Data*, pages 805–810, 2005.

[12] A. Halevy, Z. Ives, D. Suciu, and I. Tatarinov. Schema mediation in peer data management systems. In *Proc. of ICDE*, 2003.

[13] A. Y. Halevy, N. Ashish, D. Bitton, M. J. Carey, D. Draper, J. Pollock, A. Rosenthal, and V. Sikka. Enterprise Information Integration: Successes, Challenges and Controversies. In *Proceedings of SIGMOD*, pages 778–787, 2005.

[14] M. Lenzerini. Data integration: A theoretical perspective. In *PODS 2002*, pages 233–246, 2002.

[15] J. Madhavan, P. A. Bernstein, P. Domingos, and A. Halevy. Representing and Reasoning about Mappings between Domain Models. In *Proc. AAAI 2002*, 2002.

[16] A. Maedche, B. Motik, N. Silva, and R. Volz. MAFRA - A Mapping Framework for Distributed Ontologies. In *Proceedings of EKAW 2002*, 2002.

[17] D. McDermott. The Planning Domain Definition Language Manual. Technical Report 1165, Yale Computer Science, 1998. (CVC Report 98-003).

[18] D. McDermott and D. Dou. Representing Disjunction and Quantifiers in RDF. In *Proceedings of International Semantic Web Conference 2002*, 2002.

[19] R. Pottinger and A. Levy. A scalable algorithm for answering queries using views. In *Proceedings of the 26th VLDB Conference*, pages 484–495, 2000.

[20] E. Rahm and P. A. Bernstein. A survey of approaches to automatic schema matching. *VLDB J.*, 10(4):334–350, 2001.

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

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

[23] L. Stojanovic, N. Stojanovic, and R. Volz. Migrating data-intensive web sites into the semantic web. In *Proceedings of the 2002 ACM Symposium on Applied Computing*, pages 1100–1107, 2002.