

Semantic Enhancement for Enterprise Data Management

Li Ma¹, Xingzhi Sun¹, Feng Cao¹, Chen Wang¹,
Xiaoyuan Wang¹, Nick Kanellos², Dan Wolfson², and Yue Pan¹

¹ IBM China Research Laboratory

ZhongGuanCun Software Park #19, Beijing 100094, China

{malli, sunxingz, caofeng, chwang, wangxyxy, panyue}@cn.ibm.com

² IBM Software Group, 11400 Burnet Rd. Austin, TX 78758-3415, USA

kanellos@ca.ibm.com, dwolfson@us.ibm.com

Abstract. Taking customer data as an example, the paper presents an approach to enhance the management of enterprise data by using Semantic Web technologies. Customer data is the most important kind of core business entity a company uses repeatedly across many business processes and systems, and customer data management (CDM) is becoming critical for enterprises because it keeps a single, complete and accurate record of customers across the enterprise. Existing CDM systems focus on integrating customer data from all customer-facing channels and front and back office systems through multiple interfaces, as well as publishing customer data to different applications. To make the effective use of the CDM system, this paper investigates semantic query and analysis over the integrated and centralized customer data, enabling automatic classification and relationship discovery. We have implemented these features over IBM Websphere Customer Center, and shown the prototype to our clients. We believe that our study and experiences are valuable for both Semantic Web community and data management community.

Keywords: Master Data Management, Semantic Query, Mapping, Reasoning.

1 Introduction

With the increase of market competition, modern companies become more and more seriously dependent on their information, such as customer and product data. A typical situation that companies are facing is information incomplete and inconsistent across many systems, such as one product having different codes and descriptions in various markets, and one customer having different IDs in various systems. This situation mainly results from the lack of global standards (or insufficient application of standards), and the fact that data is captured many times, in many different systems. Therefore, it is critically important to keep a single truth of core entities across various systems within an enterprise to improve business efficiency.

Core business entities that a company uses repeatedly across many business processes and systems are called master data [7], such as lists or hierarchies of customers, accounts, products. Customer data is the most important kind of master data, and customer data management is becoming critical for modern enterprises because it maintains a single, complete and accurate record of customers across the enterprise

[6]. Recently, major data management solution providers, such as IBM, Oracle and SAP, have released their master data management (MDM) solutions [2][3][4], especially on customer data management (CDM, also called customer data integration, CDI) and product information management (PIM). But it does not mean that all problems in MDM have been well solved and these commercial systems meet all customers' requirements. In fact, it is reported in [6][7] that there are still some technical challenges in MDM, for instance, enterprise-wide master data models, federation and identity management. In our previous work on PIM [14], we proposed to build an expressive and flexible ontology to represent the semantics of product information, which dynamically varies with business changes. The proposed approach enables automatic categorization and relationship discovery for product data.

Different from PIM, mature industry models for customer data have been developed [8], such as the customer model for insurance and the patient model for health-care. The CDM system can directly make use of these industry models without changes or with few changes. Also, there are no strong business needs to change the customer data model after it is deployed. So, based on the industry models, the CDM system can develop well-optimized database schema and thus run in an operational way. IBM Websphere Customer Center (WCC) V6.5 [4] is such an example. It adopts different data models for different industries, uses an object-oriented schema for storage and provides more than 500 business services to manage customer data. Integrating customer data from heterogeneous data sources for a single, complete and accurate record of customers has been widely studied for decades, such as the ETL approach supported by commercial MDM solutions. In this paper, we investigate how to search and analyze the integrated customer data stored in an operational store by semantic Web technologies, instead of discussing data cleansing and integration issues. Besides customer data, the proposed method could be easily adapted to various enterprise data management solutions.

The rest of this paper is organized as follows. Section 2 introduces ontology enabled classification and relationship query. Our prototype over IBM WCC [31] is presented in Section 3, including ontology representation, mapping, SPARQL query evaluation, and reasoning. Section 4 presents experiment results and Section 5 concludes this paper.

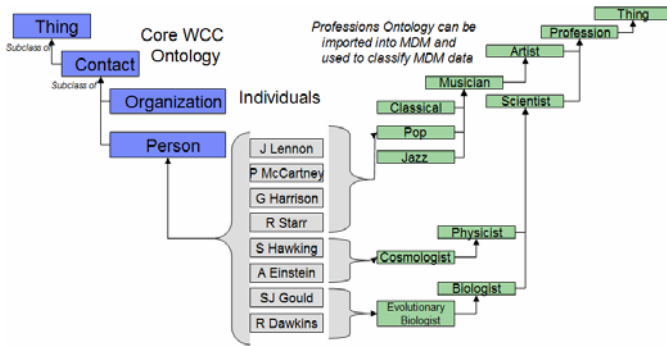
2 Semantic Query and Analysis for Customer Data Management

Based on W3C's specifications on semantic Web (such as RDFS[1], OWL[10]), we develop semantic technologies for consumption by CDM, which enables:

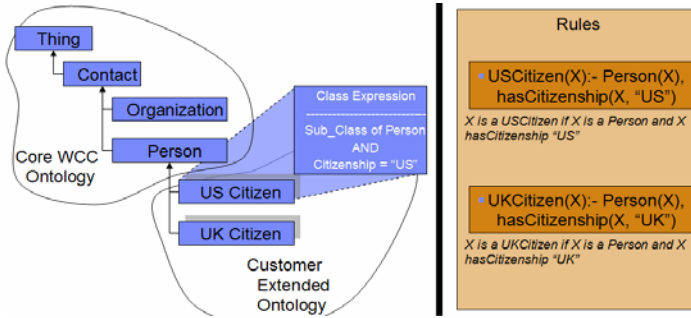
- Exposing customer data as RDF views and linking them with the open ontology/data for effective collaboration.
- Declarative definition of customer entities and relationships using logic languages supported by Semantic Web.
- Analytics to discover valuable but implicit knowledge hidden in the rich relationships among customer data.

Domain Ontology Based Classification. In reality, lots of shallow ontologies [19] (like taxonomy) for classification purpose are developed, such as profession ontology in Figure 1(a). These ontologies encoding common knowledge can be leveraged by

existing CDM systems to categorize customer data. By simply associating customer data with such domain ontologies, users can classify and retrieve their customers by profession, sector etc.. For instance, we can say "Customer A is a Cosmologist". Then, when querying all "Scientist", we will obtain Customer A. This sort of concept query is based on class inheritance in RDFS/OWL.



a) Domain ontology based classification



b) Classification based on Formal Vocabularies

Fig. 1. Semantic Query and Analysis for CDM

Classification based on Formal Vocabularies. The expressivity of OWL allows the definition of logical classes (using intersection, union and complement operators). This enables automatic classification for customer data. Also, OWL restrictions support the creation of new category of customer data. As the example in Figure 1(b) shows, we can use OWL intersection and hasValue restrictions to define a new category "US Citizen" which is a subclass of the pre-built category "Person". Using OWL reasoning, instances of this newly-defined category can be automatically retrieved. Besides OWL restrictions, we can also apply rules to formally describe new categories. An example is to define VIP customers as those whose minimum payment is larger than \$10K. That is, users can extend the core CDM vocabulary using OWL expressions or business rules, and freely define new categories at query time and classify customers using reasoning, rather than changing the deployed CDM data

model and storage model to represent new categories. This definitely helps to improve the flexibility of CDM systems. Domain ontology based classification allows users to leverage open ontologies to classify and query customers using relatively simple reasoning, whereas OWL restrictions or rules allow users to define new categories based on existing classes and properties of the core CDM ontology and use expressive ontology reasoning or rule reasoning for classification.

Relationship Discovery and Query. OWL allows the definition of richer relationships. In OWL, it is possible to define an Object Property as symmetric, functional, or transitive. OWL Object Properties are suitable to describe the complex relationships among customers, as well as those between customers and other entities like contracts. Followings are some examples of relationships in CDM: *Partner*, *Sibling*, *Trustee-Trustee*, *Owner between party and contract*, *Part-of among contracts*. Besides using OWL to define the type of relationships, we can use rules to define more general relationships in CDM. Using OWL and Datalog rule inference, we can effectively discover implicit relationships from explicit assertions. For instance, the query “Find all Insurance Contracts held by Persons having the same address as an insured property” needs to compute two implicit rule paths: *Contract->Holding->Property->Address->Person* and *Contract->Person->Address*.

Building an ontology representation for customer data (named core WCC ontology in Figure 1) will facilitate us to implement the above attractive features over existing CDM systems. Moreover, end users can query customer data via classes and relationships defined in ontologies and rules, for example “Find all Scientists living in UK” to send mailing regarding sale of lab coats. In this paper, we refer to the above features as semantic query and analysis over CDM systems, which can be realized by SPARQL queries [12]. In addition, an advantage provided by ontology and RDF is to give each CDM entity a Universal Resources Identifier (URI). That is, we can expose customer data as linked RDF data and allow effective synchronization of CDM utilities to other core business entities, such as those in PIM, or to open data like DBpedia [30].

3 System Implementation

In this section, we take IBM WCC [4] as an example to detail our implementation of semantic query and analysis for CDM [31]. But technologies developed here can be used in other CDM systems, and easily adapted to various data management solutions.

3.1 Architecture

A CDM system provides a unified customer view and an update environment to multiple channels. So, it is impractical to export all customer data into an RDF store for semantic query and analysis. Figure 2 shows an overview of the proposed system, which adds a semantic data access engine over a CDM system through a RDB-to-ontology mapping. The engine exposes a virtual RDF view for customer data.

Firstly, we create a mapping to link customer data with the OWL ontology generated and enriched from the CDM logical data model. That is, we construct a formal

ontology representation for customer data stored in the CDM system. Then, users can define business rules for knowledge discovery, and can introduce domain ontologies or OWL restrictions for classification. At query time, users can issue SPARQL queries to the semantic data access engine for data retrieval. The query can include classes and properties defined in both core and user-extended ontologies, and those in domain ontologies. A query will be parsed into a SPARQL query pattern tree. Each node in the tree will be analyzed and checked whether it implies ontology reasoning or rule reasoning. When the rule reasoning is needed, the Datalog rule evaluation will be conducted. If ontology reasoning for instance classification is required, the ontology reasoner will be called. During the reasoning process, the SPARQL query pattern tree will be modified according to the reasoning result. Finally, a SQL generator will generate a single SQL statement, which is evaluated by the database engine where the CDM store is built. In our current implementation, SOR ontology repository [17] is used to store ontologies and rules, and SHER engine [15] for ontology classification reasoning.

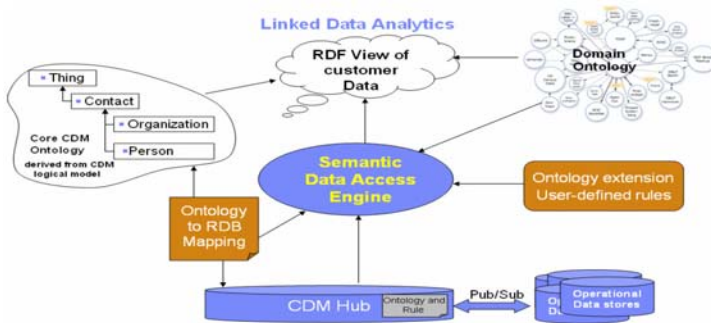


Fig. 2. The overview of Semantic CDM

Business users are able to directly use semantic query to retrieve customer data and analyze the relationships among entities involved in CDM. Another valuable use scenario of semantic query is to enable application developers to develop general analytic services over CDM systems. Now, most existing CDM systems provide only built-in services and do not allow direct SQL queries. For example, WCC provides more than 500 services for various operations and may use different optimization techniques for different services, such as materialized views. When users want additional analytic services that can be realized by the semantic queries, developers just need to write SPARQL to implement them and do not have to consider complex SQL statements and reasoning. That is, we provide a general way to develop new analytic services by SPARQL queries.

3.2 Ontology and Mapping Building

The physical storage model of CDM is similar to its logical data model pre-built for a particular industry. As introduced in the Section 1, the data model of CDM is relatively stable. The CDM's object-oriented storage model makes it possible to take full advantage of mature relational database technologies. The stable data model sacrifices

flexibility but contributes to the performance. Therefore, we should construct a relatively stable, lightweight, and coarse-grained ontology model for CDM, over which the business rules and class expressions still can be defined for expressivity. Meanwhile, we need a RDB-to-ontology mapping between the CDM storage model and the constructed ontology model so that an ontology query can be executed smoothly over the CDM system. Note that there are two types of ontologies in our system. The first type of ontology is a semantic representation of underlying customer data and derived from the logical model of the CDM system. The second type of ontology is the imported domain ontology, which captures the domain knowledge and enables users to view customer data based on the domain vocabulary.

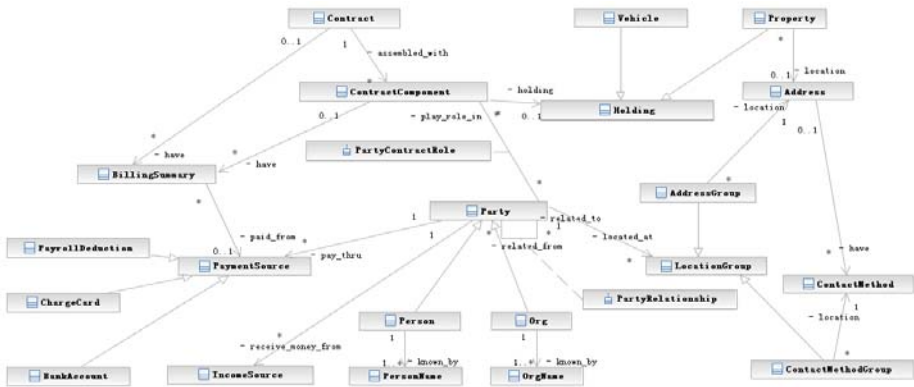


Fig. 3. Part of High Level Industry Model for CDM

CDM Data and Storage Model. Figure 3 shows part of the WCC’s data model, in which the attributes of entities are omitted. Within the model, *Party* is the most important concept, with *Person* and *Organization* as its two subconcepts. Other entities are associated with *Party* via various types of relationships. *Contract* is another important concept in CDM, which represents the purchasing agreements of customers. In this model, most relationships are represented as entities, e.g., *Party_Relationship*, *Contract_Relationship*, *Party_Contract_Role* (linking *Party* with *Contract*). Besides the entities and relationships, there are some simple hierarchies among them. For example, *Bank_Account_Record*, *Charge_Card_Record*, and *Payroll_Deduction* are disjoint sub-classes of *Payment_Source*. The WCC’s storage model is very close to this logical data model, by considering entities as the tables, their attributes as the columns in the corresponding tables, and their relations between entities as foreign keys or tables.

Ontology Representation of CDM. In our study, we found that RDFS can capture the semantics of the CDM data model. We construct the ontology representation of the CDM model in a relatively straightforward way. For example, *Party*, *Person*, and *Organization* are defined as classes, and their hierarchy is modeled by *subclassOf* semantics. All objects stored in *Party*, *Person* and *Organization* tables are their instances. Although the CDM ontology’s expressivity is RDFS, users can define very expressive OWL classes over it.

In CDM, many N-Ary relationships are used to describe the variability of customer data. For instance, relationships can be time-dependant, such as *Party_Relationship* having attributes *Start_Datetime* and *End_Datetime*. Moreover, CDM has a set of pre-defined types for *Party_Relationship*, such as *AncestorOf*. OWL only allows the definition of binary relationships, but we need to represent N-Ary relationships in CDM. The best practices documented at [23] for N-Ary relationship representation in OWL provide some choices as a starting point. In our case, we model N-Ary relationships as classes. For example, *Party_Relationship* is an OWL class, and the specific relationship between two parties is an OWL individual and has *Start_Datetime* and *End_Datetime* as properties. The characteristics of relationships (such as symmetric, transitive) can be enforced by rules. We can use a rule engine for relationships inference, which is discussed in more detail in next sub-sections.

RDB-to-Ontology Mapping. Given the designed CDM ontology, we have to map customer data to it for semantic queries. The mapping is critical to translate a SPARQL query on ontologies into an executable SQL on customer data. In 2008, W3C has formed an incubator group to investigate the standardization of a mapping language between ontology and relational database. Here, we use D2RQ mapping language [9], which is a declarative language for defining the mapping between an ontology and a relational data model. To generate a complete mapping file, we adopt a semi-automatic method. We firstly generate an initial D2RQ mapping based on WCC's physical schema using the D2RQ mapping tool, and then revise it manually according to our ontology.

3.3 SPARQL Query Evaluation

SPARQL queries allow to access customer data, user-defined categories and imported domain ontologies. Figure 4 shows the high level workflow of our query processing. A SPARQL query is firstly parsed into a query pattern tree (shown in Figure 5), and then the Datalog rule engine checks if there are rule heads in query nodes and expands rule heads accordingly. If there is a recursive loop in expanding the rule heads, the Datalog engine will evaluate the corresponding query nodes directly and generate a temporary table as evaluation results. In this process, the Datalog engine will call the SQL generator, which is responsible to generate optimized SQL queries, to get SQL statements for rule reasoning. Accordingly, the Datalog engine updates the SPARQL pattern tree by modifying the query nodes that need reasoning. At last, the SQL generator translates the updated SPARQL pattern tree into a single SQL statement. As mentioned in Section 2, we use an ontology reasoner for instance classification. If a SPARQL query needs classification reasoning (i.e. the triple pattern using *rdf:type* as predicate), the ontology reasoner will be invoked and the results are temporarily stored in a *TypeOf* table, which contains user-defined classification information as well (namely user-specified *TypeOf* information through domain ontologies). That is, the query nodes with *rdf:type* as predicate are translated into a sub-query over a *TypeOf* table.

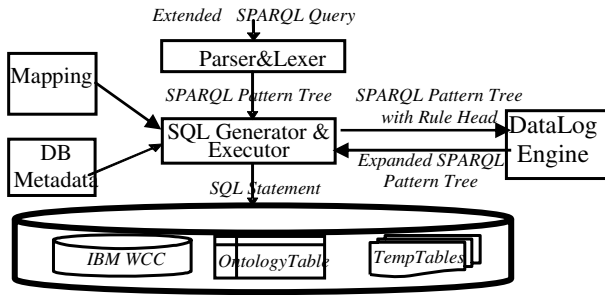


Fig. 4. The Workflow of Query Processing

Some translation methods have been proposed to translate a SPARQL query into SQL queries for evaluation based on the RDB-to-ontology mapping, such as D2R [20], Virtuoso’s SPARQL to SQL translation [28], etc.. But these approaches may generate inefficient SQL queries in some cases. Our method makes use of mapping and all other relevant information (such as database catalog) to optimize query generation. To fully utilize the well-developed SQL optimization engine, our method translates a SPARQL query into a single SQL statement. The generated SQL query can be directly embedded into other CDM SQL queries as a sub-query, which provides a way to integrate normal CDM queries with semantic queries.

As introduced earlier, there are some N-Ary relationships in CDM. Unfortunately, the current SPARQL specification does not directly support N-Ary relationships. Although SPARQL may express N-Ary relationships by combining many triple patterns, it is very inconvenient. Therefore, we extend the SPARQL query with a new pattern, called N-ARY pattern, denoting N-Ary relationships and the head predicates in the user-defined rules. The following example shows a SPARQL query with two N-ARY patterns *VIP_Organization* and *Legal_Relationship*, which finds all persons with their addresses (if available) who became the legal persons of VIP organizations that have Party relationship with IBM or HP after year 2000. Also, we show rules used to answer this query.

```

SELECT ?person ?address
WHERE {
    ?org1 wcc:PartyRelationship ?org2 .
    ?org2 rdf:type wcc:Organization .
    VIP_Organization(?org1).
    Legal_Relationship (?person, ?org1, ?year).
    OPTIONAL {?person wcc:address ?address} .
    {
        {?org2 wcc:Org_Name "IBM"} UNION
        {?org2 wcc:Org_Name "Hp"}
    } .
    FILTER (?year > "2000-1-1" )

r1. Subs_Org_Relationship(x, y) :-
wcc:Organization(x), wcc:Organization(y), wcc:PartyRelationship(z),
wcc:Related_From(z, x), wcc:Related_To(z, y),
wcc:PartyRelationshipType(z, "Subsidiary") ;;

r2. Sub_Org_Relationship(x, y) :-
Subs_Org_Relationship(x, z), Subs_Org_Relationship(z, y);;.

r3. Legal_Relationship(x, y, z) :-
wcc:Person(x), wcc:Organization(y), wcc:PartyRelationship(u),
wcc:Related_From(u, x), wcc:Related_To(u, y),
wcc:Start_Time(u, z), PartyRelationshipType(u, "Legal Person");;.

r4. Legal_Relationship(x, y, z) :-
Legal_Relationship(x, u, z), Sub_Org_Relationship(u, y);;.

r5.VIP_Organization(x):-
wcc:Organization(x), wcc:Client_Importance(x, "High");;.
    
```

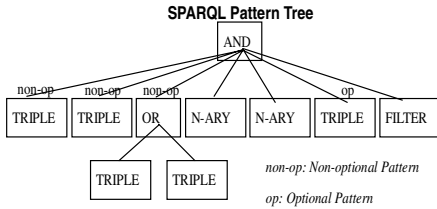



Fig. 5. A SPARQL Pattern Tree

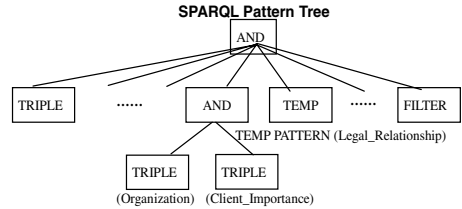


Fig. 6. Rule Expansion on A Pattern Tree

Generally, the graph pattern of a SPARQL query can be expressed as a SPARQL pattern tree, which is introduced in our previous work [16]. The pattern tree shows the backbone of the SPARQL query and is very useful for the SPARQL-to-SQL translation. Figure 5 shows the SPARQL pattern tree of the above example query. There are five types of nodes in the pattern tree:

AND node: It corresponds to the conjunction of graph patterns in a SPARQL query. Each child node has a flag to indicate whether the node is optional or not. By this way, optional patterns in SPARQL are also covered by AND nodes.

OR node: It corresponds to a UNION pattern in a SPARQL query.

TRIPLE node: It represents a graph pattern with a single triple. The subject, predicate and object in the triple could be constants or variables.

FILTER node: It represents a filter expression in a SPARQL query.

N-ARY node: It represents an N-ARY pattern and often directly corresponds to a rule head. Each element in the N-ARY node could be a constant or a variable.

Due to space limitations, we highlight new structures and optimization schemes used in our mapping based translation method.

(1) *Facet of IRI/Literal pattern to translate filter expressions.* Extended from the facet of literal defined in [16], a facet of an IRI/Literal pattern is a complex SQL expression.

(2) *Semi-SQL structure to bridge the columns in SQL and the variables in SPARQL.* Semi-SQL structure consists of several sections. These sections are connected by UNION in semantics. Each section includes two fields: a) SQL field, to keep a SQL statement as the intermediate translation result. b) Mapping field, to record the mapping between each variable and IRI/Literal pattern binding.

(3) *Integrity constraint based simplification for IRI/Literal comparison.* The basic ideas are: a) to compare the IRI/Literal pattern on the raw columns instead of the generated IRIs or literals; b) to compare the constants in IRI/Literal pattern level instead of instance level. TRIPLE nodes often include some constants. Taking the constants as a special IRI/Literal pattern, the prefixes and the data types of the constants are adopted to filter out unmatched property bridges and class maps, which can effectively reduce the number of sub-queries.

(4) *Flattened SQL to detect unnecessary JOINS.* When an AND-pattern contains more than one sub-pattern, the SQL statements of the sub-patterns should be merged for one SQL. An alternative way of merging sub-SQL statements is to take one sub-SQL as a table and embed the table in the FROM clause of the other sub-SQL. However,

the unnecessary JOINS in embedded IRI/Literal patterns may lose the opportunity of being detected and removed. Therefore, we adopt a flattened SQL strategy to merge the SELECT, FROM and WHERE clauses of the sub-SQL statements, respectively. By this way, the IRI/Literal patterns can be directly generated using the columns in the new SELECT clause. Once the sub-queries from child pattern nodes are flattened, it is easy to find the unnecessary self-joins on primary key. In that case, two tables in the FROM clause can be reduced into one, and the costly join operation is avoided.

3.4 Reasoning in CDM

In this section, we first define the semantics of the reasoning system and then introduce the reasoning approach.

3.4.1 Semantics of CDM reasoning

For the better understanding of reasoning semantics, we first categorize the reasoning in CDM according to the reasoning scope as follows.

1) Reasoning for ontology: The reasoning introduced in this category is performed based on the CDM ontology. Specifically, it is either to classify data into newly-defined classes or to find implicit assertions from the CDM ontology. Under this category, we further classify the reasoning as the following two types:

Reasoning based on Description Logic (DL): We apply IBM SHER engine [15] to support all concept assertions. SHER reasoner uses a novel method that allows for efficient querying of SHIN ontologies with large ABoxes in databases. Currently, this method focuses on instance retrieval that queries all individuals of a given class in the ABox. It is well known that all queries over DL ontologies can be reduced to consistency check, which is usually checked by a tableau algorithm. Motivated by the fact that in most real ontologies: 1) individuals of the same class tend to have the same assertions with other individuals; 2) most assertions are in fact irrelevant for purposes of consistency check, we proposed to group individuals which are instances of the same class into a single individual to generate a summary ABox of a small size. Then, consistency check can be done on the dramatically simplified summary ABox, instead of the original ABox. The SHER reasoner implements this reasoning approach on top of SOR's storage component. It is reported in [15] that SHER can process ABox queries with up to 7.4 million assertions efficiently.

Reasoning based on Description Logic Programs (DLP): DLP reasoning is applied to discover the implicit role assertions (relationships) in the CDM ontology. Description logic programs (DLP) is an intersection of description logic (DL) and logic programs (LP). As a subset of LP, DLP can be fulfilled by horn rule reasoning. An example rule in DLP could be $wcc:p(x,z):-wcc:p(x,y) . wcc:p(y,z)$, where p is a transitive property in the CDM ontology. Compared with DL reasoning, the complexity of which is NEXPTIME-complete, the LP can be computed in P-time. It has been proved that DLP reasoning is sufficient for RDFS ontology. For the ontology with more expressivity, such as OWL-DL, DLP reasoning is sound but incomplete. In our application, DLP is applied in the part of ontology reasoning. The program, which consists of a fixed set of horn rules for DLP reasoning, is denoted as P_{DLP} .

2) Reasoning for user-defined rules: In our application, users can define a set of business rules (denoted as program P_{user}) on top of ontology to enrich the semantics. The rules are in the form of Datalog-safe rules [27], with the extension of aggregation functions. For example, the rules r1 to r5 in Section 3.3 are user-defined rules. Specifically, r1 and r2 define the sub-organization relationship. R3 and r4 define the legal-reprehensive relationship between a person and an origination. Also, the rule r5 defines the concept of VIP organization. An important constraint for user-defined rule is that the rule head could not be the DL predicate.

For clarity, we refer to DL reasoning as ontological reasoning and logic programs as rule reasoning. In general, the combination of ontological reasoning and rule reasoning causes the undecidability problem. In [24], Motik et al. proposed a decidable combination of OWL-DL with rules by restricting the rules to so-called DL-safe ones. In our work, we separated these two types of reasoning to gain the decidability. The semantics of our knowledge base system is as follows. First, let KB be the knowledge base corresponding to the CDM ontology. In addition, let KB' denote the extension of KB by importing domain ontology and/or defining new OWL restrictions. We apply the ontological reasoning Π on KB' to find all concept assertions, and the result model is represented as $\Pi(KB')$. Given the program $P = P_{DLP} \cup P_{user}$ (where P_{DLP} and P_{user} are the program for DLP rules and user-defined rules respectively), the fixpoint [27] of $\Pi(KB')$ can be computed as $P(\Pi(KB'))$. Any assertion α is called entailed by our reasoning system iff it is contained in $P(\Pi(KB'))$.

Note that the example in [25] shows that separating ontological reasoning and rule reasoning may cause incompleteness problem. However, due to the simplicity of the CDM ontology (RDFS), our reasoning scheme is practical for customer data management in terms of decidability and soundness, with the minor drawback on completeness.

3.4.2 Proposed solution

After defining the semantics, we introduce the high-level design of our reasoning system. In terms of the reasoning scheme, we apply a runtime-based reasoning, i.e., the reasoning is performed at query time. Compared with materialize-based reasoning, in which all the inference results are pre-computed and stored in database, despite having longer response time, the runtime reasoning occupies much less disk space and does not have the update problem. Given the large size customer data, it is obviously not practical to materialize all inference results. We have two reasoning components. For ontological reasoning, SHER engine is deployed to do domain ontology based classification and classification based on formal vocabularies and to discover all concept assertions for the CDM ontology. As another key component, a Datalog engine is designed and implemented for supporting the reasoning for all the rules.

Tree expansion approach. Recall that the functionality of the CDM reasoning system is to rewrite original SPARQL query tree, such that every node on the tree can be directly interpreted and processed by the SPARQL query engine. In other words, after processed by the reasoning system, the tree becomes ready to be translated into one SQL statement by the SQL generator component.

Technically, SHER engine will be invoked for every node corresponding concept assertion in the SPARQL query tree. For each node related to the rule head, the rule

Algorithm: TreeExpansion**Input:**

node: the root of the SPARQL query tree
P: the program for rule reasoning (the set of rules)

dGraph: dependency graph of program *P*

hashMap: the mapping between rule-head predicate and temporary table

Output:

The root of the expanded SPARQL query tree

Method:

```

if (node is not a leaf)
  for each child node chNode
    chNode = TreeExpansion(chNode, hashMap, dGraph);
  return node;
/* cases for leaf node */
if (node does not requires reasoning) return node;
if (node corresponds to a type-of assertion) /* require ontological reasoning */
  Set node based on SHER reasoning result;
  return node;
/* node is related to rule head */
if (node can be found in hashMap) /* i.e., has been evaluated before */
  node = corresponding temporary pattern in hashMap;
  return node;
else /* rule reasoning */
  if (node is not in a strong connected subgraph of dGraph) /* not recursive */
    Build sub-tree sRoot by expanding node based on related rules;
    return TreeExpansion(sRoot, hashMap, dGraph);
  else /* recursive rule evaluation */
    Find the strong connected sub-graph scGraph that contains node;
    for each node dNode that is depended by a node in scGraph
      and dNode  $\notin$  scGraph
        dNode = TreeExpansion(dNode, hashMap, dGraph);
        Call SQLGenerator(dNode) to get SQL statement;
        Create temporary table for dNode based the SQL;
        Update hashMap;
    /* call datalog evaluation for a set of rules */
    Find the set of rules ruleSet that corresponds to all nodes in scGraph;
    DatalogEvaluator(ruleSet, hashMap); // hashMap is updated
    node = corresponding temporary pattern in hashMap;
    return node;

```

Fig. 7. TreeExpansion Algorithm

reasoning component will process it based on the program *P* (i.e., the set of all rules). According to the characteristics of related rule(s), each rule-head node will either be expanded into a sub pattern tree or be replaced by a temporary pattern (corresponding to a temporary table). Specifically, if the node corresponds to one or multiple non-recursive rules¹, it can be expanded as a sub pattern tree. For example, suppose that in Figure 5, the N-Ary pattern on the left relates to a non-recursive rule *r5*. Then, we can replace the N-Ary pattern by a sub pattern tree that is built based on *r5*, as shown in Figure 6. For the node corresponding to recursive rules, we have to trigger the Datalog engine to compute the fixpoint for all related rules and append a temporary pattern to replace the node. Suppose in Figure 5 the N-Ary pattern on the right corresponds to the rule head of *r4*. Because *r4* is a self-recursive rule, the N-Ary pattern can not be

¹ A rule *r* is called recursive if it depends on itself or mutually depends on other rules. Otherwise, *r* is non-recursive rule.

expanded as a sub pattern tree. Instead, we do Datalog evaluation and create the temporary pattern for it, as can be seen in Figure 6.

Importantly, the principle of SPARQL tree rewriting is to maintain the tree structure (i.e., expand the node) as much as possible and generate the temporary pattern (i.e., trigger the Datalog evaluation) only when necessary. The aim is to give the SQL generator more information for optimization. Intuitively, the more information the SQL generator can have, the better chance exists to get the final generated SQL optimized.

TreeExpansion algorithm. To follow the above principle, we propose a key algorithm to rewrite the SPARQL query tree. Figure 7 shows the pseudo code of the algorithm. The algorithm recursively expand the query tree based on the ontological reasoning and rule reasoning. Since the SHER engine has been introduced in [27], for simplicity, in this section, we focus on the rule reasoning part. If a node in the SPARQL query corresponds to a rule-head predicate, the rule reasoning engine is triggered. First, as in traditional Datalog approach, by checking dependency graph² of program P , we can determine if the related rule(s) is recursive or not. For non-recursive rule (i.e., the rule head is not in a strong connected sub-graph³ of the dependency graph), we expand the node as a sub-tree and then recursively call *TreeExpansion* method on the sub-tree. If the rule is recursive, we compute the strong connected sub-graph *scGraph* it belongs to. Then, we search on the dependency graph to find any *dNode* such that 1) *dNode* is not contained in *scGraph* and 2) at least one node in *scGraph* depends on *dNode*. If there does not exist such *dNode*, it means that the all recursive rules corresponding to the nodes in *scGraph* do not depends on other rules. In this case, we can call Datalog evaluation for computing the fixpoint for these recursive rules. Otherwise (*dNode* exists), before the Datalog evaluation for recursive rules related to *scGraph*, we have to get the temporary table for each *dNode*. So, we treat each *dNode* as an N-Ary pattern. After recursively call *TreeExpansion* to expand *dNode*, the SQL generator is called to get the SQL statement, and the data for *dNode* is retrieved and stored in a temporary table. From the algorithm, we can see that our Datalog evaluation will call *DatalogEvaluator* only when we need to compute the fixpoint for a set of recursive rules. *DatalogEvaluator* is implemented based on the semi-naïve evaluation approach with Magic Sets optimization [27]. After the evaluation, *DatalogEvaluator* will update the temporary table *hashMap*, which is designed to avoid evaluating the same set of rules multiple times.

4 Experimental Results

We implemented our approach to semantic query and analysis over IBM Websphere Customer Center [31]. In this section, we report the experiment results to show the

² A dependency graph of a program P is defined as a pair $\langle V, E \rangle$, where V is the set of the rule-head predicates in P , and E is the set of direct edges that shows the dependent relationship among elements in V .

³ A graph G is called a strong connected graph if for every node pair $(n1, n2)$ in G , there exist a path from $n1$ to $n2$. Given a program P , a rule in P is recursive if its head predicate is in a strong connected sub-graph of DG , where DG is the dependency graph of P .

effectiveness and efficiency of the proposed approach. All experiments are conducted on a 2.8GHz Intel Pentium-D PC with 2GB RAM, running Windows XP Professional. Through a 1.0 Gbps intranet, a 2.66GHz Intel Core2 PC with 2GB RAM serves as the backend WCC server, with IBM DB2 9.1 Enterprise edition as the database.

In the experiments, we used a data set from a real WCC customer. The main entities in the data set include 1.9M contracts, 1.0M claims, 2.0M contacts (parties) and 3.2M location groups. The data set contains 1.9M records for *contact-contract* relationship. To estimate the performance of semantic queries that are of interest to WCC customers, additional synthetic data is generated to enrich the data set. We generate 1.0M records for *contact-contact* relationship, *contract-contract* relationship, *contact-claim* relationship and *contract-claim* relationship, respectively.

Since our approach translates a SPARQL query into a single SQL statement, its effectiveness can be evaluated by the correctness of the resulting SQL statements. The system performance is demonstrated in terms of query translation time, response time, and retrieval time, i.e., the duration from the time that a query is issued to the time that a SQL statement is generated, to the time that first answer is retrieved, and to the time that the last record in the result is retrieved, respectively.

Table 1 gives the sample rules used in the experiments. Note that in our test cases, all reasoning tasks are covered by the rule inference. R1 and R2 define the relationships that will be frequently used. R3 to R7 are designed for the scenario below: an insurance company wants to analyze each contract to see how many large claims (claims with amount greater than \$5000) are made from it. Table 2 gives the semantics of queries used in the experiments. Indeed, these types of queries are interested and suggested by customers. Not all sample queries are shown in Table 3 due to space limitations. Except for MQ4, all queries can be translated into a SQL statement without any inference result materialized in temporary tables. MQ2-1 and MQ2-2 have the same query semantics but MQ2-2 utilizes rule predicate (N-Ary pattern) to specify *partner-trustee* relationships. The same situation happens for MQ3-1 and MQ3-2.

Figure 8 shows the translation time, response time and retrieval time for queries MQ0, MQ1, MQ2 and MQ3. We can see that the translation time is relatively little compared with retrieval time. Comparing MQ2-1 and MQ2-2, the expansion of the rule predicate in the SPARQL introduces a marginal additional cost. However, rules can be re-used and can make SPARQL queries concise. Note that the retrieval time is less than 3 seconds for these quite complex semantic queries. This is mainly because that our query translation approach outputs near optimal SQL statements.

To answer MQ4, rules from R3 to R7 will be triggered. Note that R4 and R6 are recursive rules. So, we have to call the Datalog engine to do evaluation twice

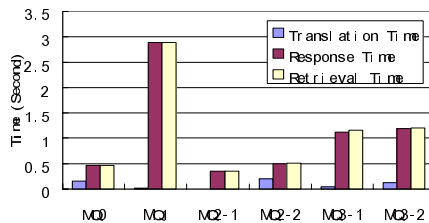


Fig. 8. Query Performance

Table 1. Sample Rules

ID	Rule
R1	Partner_Relationship(?x, ?y) :- wcc:CONTACT_Relationship_FROM(?r, ?x), wcc:CONTACT_Relationship_TO(?r, ?y), wcc:hasCONTACT_Relationship_Type(?r, ?v), wcc:Contact_Relationship_FROM_TO_NAME(?v, 'Partner');; <i>Explanation</i> : Define the partner relationship between two parties.
R2	Trustee_Relationship(?x, ?y) :- wcc:CONTACT_Relationship_FROM(?w, ?x), wcc:CONTACT_Relationship_TO(?w, ?y), wcc:hasCONTACT_Relationship_Type(?w, ?z1), wcc:Contact_Relationship_FROM_TO_NAME(?z1, 'Trustee');; <i>Explanation</i> : Define the trustee relationship between two parties.
R3	Sub_Contract(?x,?y):- wcc:CONTRACT_Relationship_FROM(?u, ?x), wcc:CONTRACT_Relationship_TO(?u, ?y), wcc:hasContract_Relationship_Type(?u, ?z), wcc:Contract_Relationship_FROM_TO_NAME(?z, 'Sub Agreement');; <i>Explanation</i> : Define the sub-contract relationship between two contracts
R4	Sub_Contract(?x,?y):-Sub_Contract(?x,?z),Sub_Contract(?z,?y);; <i>Explanation</i> : Define the sub-contract relationship is transitive
R5	Large_Claim_Agreement(?clm, ?contract, ?amount):- wcc:CLAIM_PAID_Amount(?clm,?amount), wcc:CLAIM_CONTRACT_Relationship_To(?u, ?clm), wcc:CLAIM_CONTRACT_Relationship_From(?u, ?contract); ?amount > "5000"^^xsd:double; <i>Explanation</i> : Define the relationship Large_Claim_Agreement, indicating that a contract has a claim with the amount more than \$5000
R6	Large_Claim_Agreement(?clm, ?contract, ?amount):- Large_Claim_Agreement(?clm, ?subcontract, ?amount), Sub_Contract(?subcontract, ?contract);; <i>Explanation</i> : Define that a contract is involved in the large_claim_Agreement relationship if its sub-contract has large claims
R7	Contract_NumOfLClms(?contract, ?Agg_numclm):-Large_Claim_Agreement(?clm, ?contract, ?amount);; Group by <?contract> Count(?clm) as ?Agg_numclm. <i>Explanation</i> : Compute the contract and the total number of large claims of the contract

Table 2. Meaning of Queries

ID	Query Explanation
MQ0	Find the claims that are associated to a given contract (1162960045467) and have the amount greater than \$5000
MQ1	Find the claimants of the claims that are associated to the contract 1162960045467 and have the amount greater than \$5500
MQ2	Find the person who has the partner 1000020 and is the trustee of the contact 1000045.
MQ3	Find the claimant (with their guardians) who are related to the replacement of contract with id 1000
MQ4	Find the number of large claims (amount greater than 5000) made on contract 1162960045467

(for R4 and R6 respectively) and store the inference results in temporary tables. A naïve way to answer MQ4 is to firstly to compute the number of large claims for all contracts, and then to select the record for the given contract 1162960045467. In the experiment, this naïve approach takes 40.109 seconds as the retrieval time. A smarter approach is to apply the filter condition at early stage of the query so that the inference is performed only for the relevant data. The classical Magic Sets algorithm [29] helps to realize the later approach by rewriting the rules based on the filter condition. To improve the query performance, we implemented the Magic Sets algorithm in our system. As a result, the retrieval time for MQ4 is improved to 6.453 seconds.

We also tried to run MQ0, MQ1, MQ2-1 and MQ3-1 on D2R sever [9] (V0.3.2 in our tests), an open source engine for RDF access to relational databases. However,

Table 3. Sample Queries

ID	SPARQL query
MQ0	Select ?clm ?amount where { ?u wcc:CLAIM_CONTRACT_Relationship_To ?clm. ?u wcc:CLAIM_CONTRACT_Relationship_From <WCC65DB.CONTRACT/1162960045467>. ?clm wcc:CLAIM_PAID_Amount ?amount. Filter(?amount>5000)}
MQ1	Select ?contact ?clm ?amount where {?u wcc:CLAIM_CONTRACT_Relationship_To ?clm. ?u wcc:CLAIM_CONTRACT_Relationship_From <wcc65db.contract/3000>. ?clm wcc:CLAIM_PAID_Amount ?amount. ?v wcc:CLAIM_ROLE_Relationship_From ?clm. ?v wcc:CLAIM_ROLE_Relationship_To ?contact. ?v wcc:hasCLAIM_ROLE_Relationship_Type ?type. ?type wcc:CLAIM_ROLE_NAME 'Claimant'. Filter (?amount >5500)}
MQ2-1	Select ?y where { ?r wcc:CONTACT_Relationship_FROM <WCC65DB.CONTACT/1000020>. ?r wcc:CONTACT_Relationship_TO ?y. ?r wcc:hasCONTACT_Relationship_Type ?v. ?v wcc:Contact_Relationship_FROM_TO_NAME 'Partner'. ?r1 wcc:CONTACT_Relationship_FROM ?y. ?r1 wcc:hasCONTACT_Relationship_Type ?v1. ?r1 wcc:CONTACT_Relationship_TO <WCC65DB.CONTACT/1000045>. ?v1 wcc:Contact_Relationship_FROM_TO_NAME 'Trustee'};
MQ2-2	Select ?y where {Partner_Relationship(<WCC65DB.CONTACT/1000020> ?y). Trustee_Relationship(?y <WCC65DB.CONTACT/1000045>)};
MQ4	Select ?contract ?numofClaims where { Contract_NumOfClms (?contract ?numofClaims). FILTER (?contract = < WCC65DB.CONTRACT/1162960045467>)}

D2R cannot well support these queries especially when the queries may introduce a number of JOINS. This is because that D2R server provides a memory based solution and does not use enough optimizations (not optimized for graph traversals). As D2R does not support N-Ary patterns and reasoning, it cannot execute MQ2-2, MQ3-2 and MQ4.

5 Conclusions

In this paper, we presented the use of semantic Web technologies for enterprise customer data management. Taking WCC as an example, we introduced an ontology representation for industry customer data model. An effective and practical approach to SPARQL query processing was proposed, with the ontology and rule reasoning incorporated. We believe that the presented approach could be easily adapted to various enterprise data management solutions for semantic query and analysis.

References

- [1] Customer Data Integration: Market Review & Forecast for 2005-2006, A CDI Institute MarketPulseTM In-Depth Report
- [2] Oracle Data Hub (2005), http://www.oracle.com/data_hub/index.html
- [3] SAP Master Data Management (2005), <https://www.sdn.sap.com/irj/sdn/developerareas/mdm>
- [4] IBM Websphere Customer Center, <http://www-306.ibm.com/software/data/masterdata/customer/>
- [5] IBM Websphere Product Center, <http://www-306.ibm.com/software/data/masterdata/product-info/>
- [6] Gartner Reports, Magic Quadrant for Product Information Management (2006)
- [7] Morris Henry, D., Dan, V.: Managing Master Data for Business Performance Management: The Issues and Hyperion' s Solution. IDC white paper (2005)

- [8] IBM Industry Models,
<http://www-306.ibm.com/software/data/ips/products/industrymodels/>
- [9] The D2RQ Platform v0.5.1 - Treating Non-RDF Relational Databases as Virtual RDF Graphs, <http://sites.wiwiwiss.fu-berlin.de/suhl/bizer/D2RQ/spec/>
- [10] Smith, M.K., Welty, C., McGuinness, D.: OWL web ontology language guide. W3C recommendation (February 2004)
- [11] Brickley, D., Guha, R.V.: RDF vocabulary description language 1.0: RDF schema. W3C recommendation (February 2004)
- [12] Prud'hommeaux, E., Seaborne, A.: SPARQL query language for RDF W3C recommendation (January 2008), <http://www.w3.org/TR/rdf-sparql-query/>
- [13] Hayes, P., Welty, C.: Defining N-ary Relations on the Semantic Web, W3C Working Group Note (April 12, 2006)
- [14] Brunner, J., Ma, L., Wang, C., Zhang, L.: Explorations in the Use of Semantic Web Technologies for Product Information Management. In: WWW 2007, pp. 747–756 (2007)
- [15] Dolby, J., Fokoue, A., Kalyanpur, A., Kershenbaum, A., Ma, L., Schonberg, E., Srinivas, K.: Scalable Semantic Retrieval Through Summarization and Refinement. In: Proc. of AAAI 2007, pp. 299–304 (2007)
- [16] Lu, R., Cao, F., Ma, L., Yu, Y.: An Effective SPARQL Support over Relational Databases. In: Joint ODBIS & SWDB workshop on Semantic Web, Ontologies, Databases, VLDB 2007 (2007)
- [17] Ma, L., Wang, C., Lu, J., Cao, F., Pan, Y., Yu, Y.: Effective and Efficient Semantic Web Data Management over DB2. SIGMOD, 1183–1194 (2008)
- [18] Motik, B.: On the properties of meta-modeling in OWL. In: Gil, Y., Motta, E., Benjamins, V.R., Musen, M.A. (eds.) ISWC 2005. LNCS, vol. 3729, pp. 548–562. Springer, Heidelberg (2005)
- [19] Shadbolt, N., Berners-Lee, T., Hall, W.: Nigel Shadbolt, Tim Berners-Lee and Wendy Hall, The Semantic Web Revisited. IEEE Intelligent Systems 21(3), 96–101 (2006)
- [20] Bizer, C., Cyganiak, R.: D2R Server – Publishing Relational Databases on the Semantic Web. In: Proc. of ISWC (2006)
- [21] Chen, H., Wu, Z., Wang, H., Mao, Y.: RDF/RDFS-based Relational Database Integration. In: Proc. of ICDE (2006)
- [22] Cyganiak, R.: A relational algebra for SPARQL. HP-Labs Technical Report (September 2005)
- [23] The best practices of N-Ary relationships representation in Semantic web,
<http://www.w3.org/TR/swbp-n-aryRelations/>
- [24] Motik, B., Sattler, U., Studer, R.: Query Answering for OWL-DL with Rules. In: International Semantic Web Conference (2004)
- [25] Levy, A.Y., Rousset, M.-C.: Combining Horn Rules and Description Logics in CARIN. Artificial Intelligence 104(1-2) (1998)
- [26] Donini, F.M., Lenzerini, M., Nardi, D., Schaerf, A.: AL-log: Integrating Datalog and Description Logics. J. Intell. Inf. Syst. 10(3) (1998)
- [27] Ceri, S., Gottlob, G., Tanca, L.: Logic Programming and Databases. Springer, Heidelberg (1990)
- [28] Mapping relational data to rdf in virtuoso,
<http://virtuoso.openlinksw.com/wiki/main/Main/VOSSQLRDF>
- [29] Bancilhon, F., Maier, D., Sagiv, Y., Ullman, J.: Magic sets and other strange ways to implement logic programs. In: Proc. of the Fifth ACM Symposium on Principles of Database Systems (1986)
- [30] DBpedia (2009), <http://dbpedia.org/About>
- [31] Wang, X., Sun, X., Cao, F., Ma, L., et al.: SMDM: Enhancing Enterprise-Wide Master Data Management Using Semantic Web Technologies. In: VLDB, pp. 1594–1597 (2009)