An Efficient and Complete Distributed Query Answering System for Semantic Web Data
Lehigh University Technical Report LU-CSE-07–007
June, 2007

Abir Qasem¹, Dimitre A. Dimitrov², and Jeff Heflin¹

¹ Lehigh University, 19 Memorial Drive West, Bethlehem, PA 18015
{abq2, heflin}@cse.lehigh.edu
² Tech-X Corporation, 5621 Arapahoe Avenue, Suite A, Boulder, CO 80303
dad@txcorp.com

Abstract. In this work we consider the problem of answering queries using distributed Semantic Web data sources. We define a mapping language that is a subset of OWL for the purpose of aligning heterogeneous ontologies. In order to answer queries we provide a two step solution. First, given a query we identify potentially relevant sources, which we call the source selection problem. We adapt an information integration algorithm to provide a complete solution to this problem in polynomial time. Second, we load these selected sources into an OWL reasoner, and thereby achieve complete answers to the queries. Since the time to load sources is a dominating factor in performance, and our system identifies the minimal set of potentially relevant sources, it is very efficient. We have conducted an experiment using synthetic ontologies and data sources which demonstrates that our system performs well over a wide range of queries. A typical response time for a given work load of 20 domain ontologies, 20 map ontologies and 400 data sources is just a little over 1 second.

1 Introduction

The Semantic Web provides an infrastructure that has the potential to transform the Web to a true global knowledge medium. Ontologies, expressed in a standard logic language with formal semantics, can be used in concert with web data in order to develop powerful query systems. However, the Semantic Web is a decentralized medium where different parties can and will, in general, adopt different ontologies. When many ontologies and data sources are created completely independent of one another it is quite possible that many of them will refer to the same or similar concepts. Therefore, while some of the data sources may contain data described directly in terms of a given query ontology, others may not.

To provide a unified view to the user we need a mapping mechanism to serve as a “glue” between these separately evolved, syntactically different yet semantically similar concepts. In our solution, we consider axioms of OWL ontologies (a subset of OWL to be precise) to describe a map for a pair of ontologies. The choice of OWL to articulate the alignments make these maps shareable via the Web, where any one can create
alignments and publish them in OWL for others to use. Such maps may be created manually or by using state-of-the-art ontology alignment tools [4]. We hypothesize that a web-like alignment framework will enable integration to be an emergent property of the Web. Furthermore, existing OWL tools can be used to process these maps. We note that we will not have alignments between all pairs of ontologies, but it should be possible to compose an alignment from existing alignments.

Once we have the maps between the ontologies established, we need to use them to translate the query in terms of the data sources that are available. Database researchers have developed information integration algorithms that address similar problems. In our example described in the previous section, if we consider O1 to O3 as database schemas that define the data in the databases S1 to S5, we can see a similarity between the problems addressed in the two areas. Therefore we use database information integration techniques as a starting point in solving the DQA problem in the Semantic Web.

The input of a typical information integration system for databases consists of a query and some rules relating various views (named queries) and actual relations. The system then computes a query expression that is over the actual relations. Finally the query is issued to the appropriate relation(s). The schema of the query is called a mediated schema. Most information integration techniques base their formalism on the datalog [20] data model and syntax. In datalog, the relations are denoted as predicates and the columns are referred to by the position of the arguments. For example, if p is a predicate symbol, then in p(X,Y,Z), X will refer to the first component of some tuple.

There are two main approaches in information integration that relates sources to a query. The first approach known as Global-as-View (GAV), expresses the mediated schema relations as a set of views over the data source relations [7]. For example, the relationship between a mediated Movie schema and two source relations S1 and S2, can be expressed as Movie (t,d,y,g) :- S1 (t,d,g), S2 (t,d, y). Query reformulation is easy since it amounts to view unfolding. However, the addition of new sources will require the mediated schema to change. Therefore it works best when there are a few, stable, data sources that are well known to the system. For example, integration of a corporate database is a good candidate for a GAV approach.

The second approach known as Local-as-View (LAV) expresses the source relations as views over the mediated schema [15]. For example, we may say S3 (t,y) ⊆ Movie(t,d,y, “Mystery”)\(^1\) to relate S3 to the mediated Movie schema. Note: that inclusion (as opposed to definition in GAV) denotes that S3 has some information about the Movie relation i.e. only mystery movies. LAV works best when there are many, relatively unknown data sources and there is a possibility of addition/deletion of sources. For example, a shopping agent is a good candidate for a LAV approach. Although there are relative advantages and disadvantages of both the LAV and GAV approaches, it is important to note that in either approach, a single mediated schema is difficult to achieve when many parties are involved. So their dependence on a mediated schema does not make these approaches suitable for a highly distributed medium like the Web, where it will be impossible to design a single mediated schema.

\(^1\) In the literature, LAV rules are often written using the “:-” symbol used for GAV rules. To avoid confusion, we follow the convention of Halevy et al. [13]
In our work we adapt a more flexible algorithm by Halevy et al. known as the PDMS reformulation algorithm [13]. The PDMS is a decentralized and extensible information integration architecture, in which any user can contribute new data, schema information, or even mappings between other peers’ schemas. PDMS extends the two most well known information integration approaches (LAV and GAV), replacing their single mediated schema with an interlinked collection of mappings between peers’ individual schemas. Since in the Semantic Web we can and will have queries in any ontology, we need to provide a mechanism that does not depend on a single mediated schema. Therefore PDMS’s “any schema” approach makes PDMS suitable for adaptation in the Semantic Web. Hereinafter we refer to PDMS algorithm simply as the PDMS.

Our design choices were influenced by the following observations about the current state of the Semantic Web.

– **Observation 1:** The majority of Semantic Web data is in flat RDF/OWL files as opposed to databases. Therefore, any system that uses this data must load the whole source as opposed to issuing queries to the sources as is done in PDMS. Furthermore, these files are served via HTTP.

– **Observation 2:** Many data sources may commit to the same ontology and a majority of the data commits to simple ontologies like FOAF, Dublin core, etc. As data significantly outnumbers ontologies, our focus is on instance retrieval as opposed to ontological queries. We also focused on reasoning over simple ontologies given their dominance in the Semantic Web.

– **Observation 3:** The data may change frequently but ontologies are relatively static. This means a centralized knowledgebase approach would be ineffective, since the quality of answers depends on the frequency of crawling and loading data.

This work is an extension of our initial work presented in Dimitrov et al. [6]. The previous work was based on a two-tier ontology architecture where integration was between pairs of ontologies and alignment could not be composed. Our enhancements now enable us to support a multi-tier ontology architecture. Specifically we make the following four technical contributions in this paper.

1. We identify a subset of OWL that is compatible with the mapping formalisms used by the PDMS and provide an extension for describing source relevance.
2. We formally define a source selection problem for the Semantic Web. This is the problem of identifying all potentially relevant sources, given a set of ontologies, alignments and summary meta data about sources.
3. We modify the PDMS algorithm and show that this modification serves as a complete source selection algorithm for our sublanguage of OWL. This algorithm has polynomial time complexity.
4. We demonstrate that by loading the sources selected by our algorithm into a sound and complete reasoner, we effectively can achieve sound and complete reasoning over the entire set of data sources.

Before going in to the details, we provide a motivating example to highlight the potential benefits of performing DQA in the Semantic Web. This will serve as a running example through out the paper. In the example we use description logic (DL) syntax
(unless explicitly stated otherwise) to describe various axioms of ontologies. We also use the common ontology:term syntax to indicate the ontology that defines the term.

It is conceivable that a computer retailer ontology (O1) may be mapped with a monitor supplier ontology (O2). Businesses belonging to either category will have an incentive to establish maps between their ontologies. We can create an ontology M12 that contains DL axioms that relate terms of O1 and O2. For example:

\[
\text{M12: } \text{O1: CinemaDisplay } \sqsubseteq \text{ O2: LCD } \sqcap \exists \text{ screen.\{Big\}}.
\]

This states that Cinema Displays in O1 are classified by O2 as LCDs that have big screens. The effect of having M12 is that, now if some one asks for information about LCDs, in addition to getting information from O2 sources they can also get information from O1 sources. The true power of information integration techniques however, comes from their ability to compose maps. Our intuition of maps implies a pair wise mapping between the concepts of two ontologies. But obviously we will not have mapping between all pairs of ontologies. However it should be possible to compose a map from existing maps by traversing through a “semantic connection” of maps. Consider an ontology for a user forum that reviews monitors (O3). Assume that there is an ontology M32 that contains the axiom:

\[
\text{M32: } \text{O2: LCD } \sqcap \exists \text{ screen.\{Big\} } \sqsubseteq \text{ O3: BigMonitor}
\]

This axiom implicitly establishes a mapping between O3 and O1. Now a user asking information about BigMonitors can get data from sources described in terms of O2:LCD or O1:CinemaDisplay.

Note: these two maps can be rewritten in LAV and GAV as follows

\[
\text{M12: } \text{O1: CinemaDisplay } (x) \subseteq \text{ O2: LCD } (x), \text{ screen } (x, \text{ big}),
\]

\[
\text{M32: } \text{O3: BigMonitor } (x) :- \text{ O2: LCD } (x), \text{ screen } (x, \text{ big})
\]

The rest of the paper is organized as follows: In Section 2 we formally define the source selection problem. In Section 3, we describe our mapping language that is used for information integration in the Semantic Web. In Section 4, we describe the details of our OBII system. In Section 5, we describe some experiments that we have conducted to evaluate our system. In Section 6, we compare some related work with our approach and in Section 7 we conclude and discuss future work.

2 The Source Selection Problem

Our two step approach to solving the DQA problem is inspired by Observation 1 mentioned in Section 1. In this section we formally define the first aspect of the problem which is to select the potentially relevant sources for a given query.

Based on Observation 2, we assume there are two types of information on the Semantic web: ontologies, which define terminology, and data sources, which use the terminology of ontologies to describe individuals. For our purposes, any OWL document that contains a description of a class or a property is an ontology; otherwise, the document is a data source. In Description Logic (DL) terminology the data sources provide the ABox. Each data source commits to an ontology, meaning that it only uses terms from that ontology. In OWL, this is represented by owl:imports.

**Definition 1** A Semantic Web Space SWS is a tuple \( \langle O, S \rangle \) where \( O \) is a set of ontologies, \( S \) is a set of data sources and \( S \cap O = \emptyset \).
We use a model theoretic approach for our semantics. We assume that all ontologies in the Semantic Web Space share a common domain. Since OWL DL closely corresponds to the DL SHOIN(D), we define models of ontologies and data sources accordingly. An interpretation I is a model of a Semantic Web Space SWS = (O, S) iff I is a model of every O_i ∈ O and a model of every S_i ∈ S. Given this notion of models, entailment is defined in the usual way. Note, we have intentionally chosen not to address issues of inconsistent ontologies and of many-to-many domain mappings unlike the approach taken in [2].

In order to align heterogeneous ontologies, we introduce the notion of map ontologies. These are like any other OWL ontology except they consist solely of axioms that relate concepts from one ontology to concepts of another ontology. We use the term domain ontologies for all other ontologies.

**Definition 2** A conjunctive query has the form \( H(\overline{X}) : \leftarrow B_1(\overline{X}_1), \ldots, B_n(\overline{X}_n) \) where \( \overline{X} \) are variables or individuals and each \( B_i \) is a unary or a binary atom representing a concept or role term respectively.

Our query language is based on the conjunctive query language for DLs that has been proposed by Horrocks et al. [14]. This query language overcomes the inadequacy of description logic languages in forming extensional queries. Furthermore, it corresponds to the most common SPARQL queries. Variables that appear in the head of a query are called distinguished variables and describe the form of a query’s answers (all other variables in the query are called non distinguished variables). We refer to the left hand side of \( \leftarrow \) as the head of the query and the right hand side as the body of the query.

**Definition 3** Given a Semantic Web Space SWS, an answer set \( A \) to a query \( Q \) with a body \( B \) is the set of all substitutions \( \theta \) for all distinguished variables in \( Q \) such that:

\[
\text{SWS} \models B_1 \theta \land \text{SWS} \models B_2 \theta \land \ldots \land \text{SWS} \models B_n \theta
\]

Due to Observation 3, it is infeasible to load all data sources. Therefore, our problem requires some means of specifying meta data for sources. In order to ignore sources that will not contribute to \( A \) we introduce the concept of source relevance in our framework. First we define a relation SrcInst: \( \mathcal{S} \times \mathcal{C} \to 2^D \) that maps the set of sources \( \mathcal{S} \) and the set of classes \( \mathcal{C} \) to the power set of individuals \( D \). SrcInst gives us the set of individuals of a given class in a given data source, i.e. for each \( S_i \in \mathcal{S} \) and \( C_i \in \mathcal{C} \), the interpretation of SrcInst\((S_i, C_i) = \{a \mid S_i \text{ contains } a : C_i \} \).

A “REL” statement allows us to make assertions about the type of information that a source contains so that we can ignore sources that we are certain will not contribute to new answers for a given query. The REL statement has the form REL\((S_i, C_i, CE) \) where CE is a class expression.

**Definition 4** We say source \( S_i \) conforms to a set of REL statements \( \mathcal{R} \) iff

1. for each \( r \in \mathcal{R} \) s.t. \( r = \text{REL}(S_i, C_j, CE_j) \), SrcInst\((S_i, C_j) \) is \( \neq \perp \) and SrcInst\((S_i, C_j) \subseteq CE_j \)
2. for all \( a \in C_j \) of source \( S_i \) there exists a \( r \in \mathcal{R} \) s.t. \( r = \text{REL}(S_i, C_j, CE_j) \) and \( \{a\} \subseteq CE_j \).
Note: REL statements for properties can be defined in a similar fashion.

**Definition 5** Given a Semantic Web Space SWS = ⟨O, S⟩, a set of REL statements R s.t. all S_\_i ∈ S conform to R, and a conjunctive query Q: a source S_\_i is potentially relevant iff

1. there exists an S′ and R′ s.t. each S_\_j ∈ S has a corresponding element S_\_j′ and equivalent REL statements in R′ where S′ conforms to R′ and
2. the answer set A′ for query Q on a hypothetical SWS′ = ⟨O, S′⟩ is a strict superset to the answer set A− for query Q w.r.t SWS− = ⟨O, S−⟩ where S− is S′ except S_i

**Definition 6** Given a Semantic Web Space SWS, a query Q and a set of REL statements R, the source selection problem is to identify all potentially relevant sources.

Note: If a system produces the answer set for all possible queries and Semantic Web Spaces it is a complete system. We also note that the semantics of these definitions reflect the state of the Semantic Web at any point in time. The definitions however can be extended to reflect a more dynamic Web with an approach similar to transaction semantics used in the database area. We also note that at this point we are not addressing the issue of inconsistency and trust and as such our definition does not reflect these issues.

### 3 Mapping Language

In this section we describe OWLII, our mapping language for describing maps and sources in the Semantic Web. This language is influenced by the PDMS algorithm, which uses LAV and GAV rules to describe maps and data sources. We want to define our mapping language so that it is a subset of OWL DL that is compatible with LAV and GAV. Since OWL DL is decidable, this subset will also be decidable.

#### 3.1 Describing Peer Maps

We start with Description Horn Logic (DHL) [8], which is contained in the intersection of Description Logic and Horn Logic. More specifically, it is a fragment of OWL corresponding to Horn clauses, i.e. implications with a set of conjuncts in the body and at most one literal in the head. We observe that GAV rules are essentially equivalent to Horn clauses without function symbols. Thus if we were using a GAV information integration system, DHL would be a suitable Semantic Web mapping language.

Although DHL is a good starting point, the PDMS query reformulation algorithm also supports LAV rules, and as such we can use a language that is more expressive than DHL. Since LAV rules are essentially view inclusions (i.e. they state that the view is contained by some set of schema sub goals) a rule of the form B (X) ⊆ H_1 (X_1), H_2 (X_2), …, H_m (X_m) can be written in First Order Logic (FOL) as an implication with a single antecedent and multiple consequents as follows: B (X) → H_1 (X_1) ∧ H_2 (X_2) ∧ … ∧ H_m (X_m) where B (X) and H (X) refer to FOL atoms with a vector of X variables.

6
We observe that it is possible to represent any rule of the form $B_1(\mathbf{X}_1) \land B_2(\mathbf{X}_2) \land \ldots \land B_n(\mathbf{X}_n) \rightarrow H_1(\mathbf{X}_1) \land H_2(\mathbf{X}_2) \land \ldots \land H_m(\mathbf{X}_m)$ as a pair of GAV and LAV rules using a simple rewriting technique that involves the introduction of a fresh predicate. Let $\mathit{fp}$ be a fresh predicate that appears nowhere else in the system and let $\mathit{fp}(\mathbf{X}) = B_1(\mathbf{X}_1) \land B_2(\mathbf{X}_2) \land \ldots \land B_n(\mathbf{X}_n)$. The axiom $B_1(\mathbf{X}_1) \land B_2(\mathbf{X}_2) \land \ldots \land B_n(\mathbf{X}_n) \rightarrow H_1(\mathbf{X}_1) \land H_2(\mathbf{X}_2) \land \ldots \land H_m(\mathbf{X}_m)$ is then equivalent to the following pair of axioms: $B_1(\mathbf{X}_1) \land B_2(\mathbf{X}_2) \land \ldots \land B_n(\mathbf{X}_n) \rightarrow \mathit{fp}(\mathbf{X})$ and $\mathit{fp}(\mathbf{X}) \rightarrow H_1(\mathbf{X}_1) \land H_2(\mathbf{X}_2) \land \ldots \land H_m(\mathbf{X}_m)$. These correspond to the GAV rule of the form $\mathit{fp}(\mathbf{X}) :- B_1(\mathbf{X}_1) \land B_2(\mathbf{X}_2) \land \ldots \land B_n(\mathbf{X}_n)$ and LAV rule of the form $\mathit{fp}(\mathbf{X}) \subseteq H_1(\mathbf{X}_1), H_2(\mathbf{X}_2), \ldots, H_m(\mathbf{X}_m)$. Therefore, any rule in our general form described above can be written as either GAV, LAV or a LAV/GAV pair.

Thus using LAV rules as well as GAV rules allows us to be more expressive then DHL by adding existential restrictions on the right hand side of a concept inclusion. This restriction translates in FOL to an implication with multiple consequents that use the same existentially quantified variable. This was not part of DHL as this could not be converted to Horn. However, this is possible in OWLII because multiple consequents are allowed. Furthermore, since existential restrictions can appear on both sides of a concept inclusion, they can also, appear on both sides of an equivalence.

Following the manner of Grosof et al. [8] we define the subset of DL for OWLII.

**Definition 7** $L_{ac}$ is a DL language where $A$ is an atomic class, and if $C$ and $D$ are classes and $R$ is a property, then $C \sqcap D$ and $\exists R.C$ are also classes.

**Definition 8** $L_a$ includes all classes in $L_{ac}$. Also, if $C$ and $D$ are classes then $C \sqcup D$ is also a class.

**Definition 9** $L_c$ includes all classes in $L_{ac}$. Also, if $C$ and $D$ are classes then $\forall R.C$ is also a class.

**Definition 10** We now define a OWLII map ontology as a set of OWLII axioms of the form $C \sqsubseteq D$, $A \equiv B$, $P \sqsubseteq Q$, $P \equiv Q$, $P \equiv Q^-$, where $C$ is an $L_{ac}$ class, $D$ is an $L_c$ class, $A, B$ are $L_{ac}$ classes and $P, Q$ are properties.

In other words, OWLII map statements allow the following OWL constructs:

1. An owl:equivalentClass statement can have named classes, complex classes using owl:intersectionOf, and restricted classes using owl:someValuesFrom in either its subject or its predicate.
2. An rdfs:subClassOf statement can have all of the constructs available for owl:equivalentClass and in addition can have complex classes using owl:unionOf in its subject and restricted classes using owl:allValuesFrom as its object.
3. owl:equivalentProperty or rdfs:subPropertyOf can only have named properties in either its subject or its predicate
4. owl:inverseOf can only have named properties in either its subject or its predicate.
Note: Although OWLII can contain all the DHL axioms, in definition 10 we have chosen to omit a few. For example we have omitted transitive property. Although it will be easy to express it in OWLII it is meaningless from a mapping perspective.

In Table 1, we give a translation function $\mathcal{T}$ which takes a DL axiom of the form $C \subseteq D$, where $C$ is an $L_a$ class and $D$ is an $L_c$ class, and maps it into the FOL format for OWLII. This definition expands the one for DHL [8].

<table>
<thead>
<tr>
<th>DL axioms</th>
<th>FOL</th>
</tr>
</thead>
<tbody>
<tr>
<td>$\mathcal{T}(C \subseteq D)$</td>
<td>$\mathcal{T}_c(C, y) \rightarrow \mathcal{T}_c(D, y)$</td>
</tr>
<tr>
<td>$\mathcal{T}_a(A, x)$</td>
<td>$A(x)$</td>
</tr>
<tr>
<td>$\mathcal{T}_c((C \cap D), x)$</td>
<td>$\mathcal{T}_c(C, x) \land \mathcal{T}_c(D, x)$</td>
</tr>
<tr>
<td>$\mathcal{T}_a((\exists R.C), x)$</td>
<td>$R(x, y) \land \mathcal{T}_c(C, y)$</td>
</tr>
<tr>
<td>$\mathcal{T}_c((\forall R.C), x)$</td>
<td>$R(x, y) \rightarrow \mathcal{T}_c(C, y)$</td>
</tr>
<tr>
<td>$\mathcal{T}_a(A, x)$</td>
<td>$A(x)$</td>
</tr>
<tr>
<td>$\mathcal{T}_a((C \cap D), x)$</td>
<td>$\mathcal{T}_a(C, x) \land \mathcal{T}_a(D, x)$</td>
</tr>
<tr>
<td>$\mathcal{T}_a((\exists R.C), x)$</td>
<td>$R(x, y) \land \mathcal{T}_a(C, y)$</td>
</tr>
<tr>
<td>$\mathcal{T}_a((\forall R.C), x)$</td>
<td>$R(x, y) \rightarrow \mathcal{T}_a(C, y)$</td>
</tr>
<tr>
<td>$\mathcal{T}_a((C \cup D), x)$</td>
<td>$\mathcal{T}_a(C, x) \lor \mathcal{T}_a(D, x)$</td>
</tr>
</tbody>
</table>

Table 1. Application of $\mathcal{T}$ to DL axioms

This table can also be used for class equivalence by translating the equivalence into two subclass axioms. The property translations are straightforward. We can process any FOL formula that results from translation as follows: if it has one consequent it is GAV, if it has one antecedent it is LAV otherwise it is a LAV/GAV pair by using the rewriting technique from above.

### 3.2 Describing Sources

In Section 2, we introduced the notion of a REL statement. This plays the role of source descriptions as used by the the PDMS algorithm. If a source can express that it has relevant information, we can choose to query it as opposed to other sources that do not express this information. In this way we can locate the desired information without querying every possible source. Having relevant information, however, does not mean that the source is capable of answering the query completely. It just says that the source may have some useful information on the query.

In OWL we can express the REL statement as follows. We introduce four new predicates in a new name space "meta". They are meta:RelStatement, meta:contained, meta:container and meta:source. The meta:RelStatement encapsulates REL statement. The meta:contained, and the meta:container represent the $C_i$ and CE of the REL statement. The meta:source property represent $S_i$. Consider the following REL statement as an example. REL(http://sourceURL, Electronics, CinemaDisplay $\sqcap \exists$ madeBy:"DELL") can be expressed in OWL as follows:

```xml
<meta:RelStatement>
  <meta:source rdf:resource="http://sourceURL"/>
</meta:RelStatement>
```
Note: We needed the predicate meta:contained to explicitly state which predicate is actually used in the data. For example, if we had just made a relevant statement about cinema displays that are made by Dell, it is not clear if we have relevant information about cinema displays or things made by Dell or both. The above REL statement can be seen as a special case of OWLII LAV map where both the sides are defined using the same ontology. O1:Electronics (X) ⊆ O1:CinemaDisplay (X), O1:madeBy (X, “DELL”).

We end this section with the following observations about our source selection framework. First, although a source has relevance expressed in terms of one ontology it may be relevant to queries formulated using another ontology. Second, when the data sources contain a large amount of data, we may be able to use various data mining techniques [11] to discover relevance statements.

4 OBII: A Semantic Web DQA System

In this section we describe the details of OBII. The heart of OBII is our adaptation of the PDMS algorithm. The key aspect of our adaptation is that we use the PDMS to solve a source selection problem as opposed to the query reformulation problem that it was originally designed for. After our adapted algorithm selects the sources that have the potential to contribute to the answer set of a query, we load these sources into a sound and complete Description Logic reasoner to infer answers.

In this section we first give an overview of the PDMS algorithm. We then present our modified algorithm and argue why it is a complete source selection algorithm. Finally, we briefly discuss the implemented architecture of OBII to give the readers a flavor of the working system.
### 4.1 Source Selection Algorithm

Our algorithm is based on the PDMS algorithm, which takes as input a query and a set of views describing the sources and the maps and computes a reformulation strictly in terms of the sources. As long as there are no cycles in the maps, the PDMS algorithm computes complete reformulations in polynomial time [13]. The algorithm constructs a “rule-goal” tree: where goal nodes are labeled with atoms of the peer relations, and rule nodes are labeled with maps. The tree is constructed by expanding nodes using the maps. In order to ensure termination, the algorithm does not apply maps that have already been used along the path from the root. This is why the algorithm is incomplete in the presence of cycles. However, it is still polynomial.

To expand a goal node the algorithm looks for a match in the maps for the label of that node. When a match is found a child rule node is created which is labeled with the matched peer map. The rule node is then expanded as follows: If the matched peer map is a GAV-style mapping, a new child goal node is created for each sub goal of the peer map. If the matched peer map is a LAV-style mapping, child goal nodes are formed using the MiniCon algorithm [16].

The MiniCon algorithm identifies a minimal subset of views that is required to answer a query. In the process this algorithm creates a MiniCon (Minimal Construction) description for each set of query sub goals that cover a view. For a given MiniCon description of a goal node (w.r.t. its siblings and the matched peer map) the PDMS algorithm creates a rule node with the view of the MiniCon description and creates a child goal node with the head of the view. It also marks in the rule node all of the other sub goals that are covered by the peer map.

Each goal node is also expanded using the maps relating a data source to a peer. These maps are essentially LAV-style mappings with the actual stored relations on the left hand side of an inclusion description. The PDMS algorithm combines and inter-leaves the two types of reformulation techniques: one type of reformulation replaces a subgoal with a set of subgoals, while the other replaces a set of subgoals with a single sub goal.

The reformulation is a union of conjunctive queries over the stored relations. Each of these conjunctive queries represents one way of obtaining answers to the query from the relations stored at peers. Basically the rule goal tree is an AND-OR graph and the reformulations are all of the AND-OR traversals from root to leaf.

In our adaptation of PDMS we wanted to take advantage of the semantics that are available in OWL and RDFS ontologies. This is not considered in the original PDMS which is designed for a relational data model or its more recent implementation, the Piazza system [12] which is designed for XML’s tree data model. Recall in Section 1 we made the observation that at present the most widely used ontologies in the Semantic Web are RDF(S) ontologies like FOAF, Dublin Core, etc. (Observation 2). The primary inference required by these ontologies is taxonomic reasoning. Therefore our algorithm must support domain ontologies that have class and property taxonomies.

We now describe our source selection algorithm. Note: from here on we refer to an ontology that only has classes, properties, subClassOf and subPropertiesOf axioms as a simple ontology. For our algorithm we assume that all domain ontologies are simple ontologies, mapping ontologies are described in OWLII, and sources are described us-
ing the REL formalism and only contain ABox assertions that use named classes and properties.

Given a query Q in the form described in section 2 we unfold it to give us the starting point of our rule goal tree. Now for each atom in the unfolded query we attempt to expand it using maps and source descriptions that are available to the system. By recursively continuing with this expansion until we do not have any more maps (or source descriptions) available we build a rule goal tree in a similar fashion as the basic PDMS described above. However, our expansion is different due to the presence of domain ontologies with class and property taxonomies.

We implement taxonomic reasoning with a combination of a subPreds function and a modified unification function and apply it when comparing query subgoals to view subgoals. The subPreds function is a generalization of two operations: getting sub classes of a given class and getting sub properties of a given property in a given ontology. Since classes and properties can be thought of as unary and binary predicates respectively, this is a useful abstraction. For each query goal we use a simple reasoner to find the set of subclasses (sub properties). The match function takes three arguments: n, sp, and a, where n is the goal node, sp is a list of sub predicates of the goal node and a is a map atom. The match function implements a variation of standard unification process. It attempts to unify any of the sub predicates sp of n (including n’s predicate) with a. If there is a substitution found then n unifies with a.

We considered two options for comparing query goals to view sub goals. We can either compare subclasses of the query goal to actual view sub goals or we can compare the actual query goal to super classes of the view sub goals. The former approach can compute query goal subclasses with a single search operation and it seems more natural. The latter approach seems efficient because on average a class should have more subclasses than super classes (the branching factor in super direction will often be close to one, but in the sub direction it could be very large). However, this approach will require significant additional processing for each view goal that does not match the query. We will also have to find super classes of every class that appears in some view. We will have to repeat this every time the class appears. If every leaf class appears in some view, we essentially process the entire ontology every time we attempt to match a query subgoal. Based on the above analysis we have decided to take the first approach, which is to compare subclasses of the query goal to actual view sub goals.

Our extended expansion algorithm is presented as Algorithm 1. Please note that the MapViews and SourceViews objects mentioned in the algorithm are data structures that store the maps and the source descriptions indexed by their source ontologies for efficient retrieval. The details of these objects are discussed in Section 4.2. The source ontology of a map is the left hand side (LHS) ontology of a GAV map or the right hand side (RHS) ontology of a LAV map. After we have built the rule goal tree we read off all of the unique sources from the leaves of the rule goal tree.

Recall in the basic PDMS, each AND-OR traversal from root to leaf represents one way of answering the query. The reformulation then is obtained by the union of all of these traversals. The basic PDMS does not allow cycles in the maps as this makes query answering undecidable. It prevents cycles by never expanding a goal node with a map that was already used on the path from the root to that node in question. Therefore the
Algorithm 1 OBII node expansion.

EXPAND(Node n, MapViews MV, SourceViews SV)
1: sp ← SUBPRED(n.pred, n.ont)
2: omaps ← \{m | (n.ont, m) ∈ MV\}
3: for each v ∈ omaps do
4:   if GAV(v) and MATCH(n, sp, HEAD(v)) then
5:     for each sub goal ∈ v do
6:       create an AND child goal node
7:     EXPAND(cgn, MV, SV)
8:   else if LAV(v) and MATCH(n, sp, b) for some b ∈ BODY(v) then
9:     create an OR child node cgn for the view in MCD and mark uncles
10:    EXPAND(cgn, MV, SV)
11: spmaps ← \{m | (n.ont, m) ∈ SV\}
12: for each v ∈ spmaps do
13:   if GAV(v) and MATCH(n, sp, HEAD(v)) then
14:     for each sub goal ∈ v do
15:       create an AND child goal node
16:     else if LAV(v) and MATCH(n, sp, b) for some b ∈ BODY(v) then
17:       create an OR child node cgn for the view in MCD and mark uncles

set of AND-OR traversals will not be a complete reformulation in the presence of cyclic maps.

Theorem 1 The OBII source selection algorithm is complete in that given a set of simple domain ontologies, OWLII map ontologies and sources that conform to OWLII REL statements, it will identify exactly the potentially relevant sources. Furthermore, this algorithm terminates in polynomial time.

Proof. (sketch): Observe that the original PDMS algorithm is incomplete in the presence of cycles solely due to the check that the same map is not used twice on any path from root to leaf (of course without this check, the tree could be infinite). If we omitted the check there would be goal nodes identical to goal nodes elsewhere in the tree. Any leaves of subtrees rooted at such goal nodes would be identical to leaves of the subtree rooted at the duplicated goal node. Thus they do not identify any new sources. Furthermore, since the PDMS algorithm is polynomial [13] and our algorithm only adds a test that is linear in the size of an ontology, our algorithm is polynomial.

4.2 Architecture

In this section, we briefly discuss the overall architecture of our system and its functionality. OBII functions in two distinct and asynchronous phases: an initialization and update phase and a query phase. The initialization and update phase is carried out by the OWLIIRuleProcessor and occurs during system startup or when a new data source becomes available to the system. This phase essentially manages the system’s map knowledge base MapKB. The query phase occurs each time when a query comes in and
is implemented by QueryHandler, Reformulator and AnsweringEngine. We also use KAON2 for OWL reasoning. Note: by performing the initialization once (or updates as needed) and spawning threads for each query we achieve a gain in efficiency by sharing the map knowledge base across queries. Figure 1 shows the architecture of our system.

![OBII architecture diagram](image)

**Fig. 1.** OBII architecture diagram

**OWLIIRuleProcessor:** This module loads the OWLII peer maps and source descriptions (i.e. the REL Statements described earlier) into OBII. The peer maps are described using standard OWL syntax. They are parsed into the system as a MapView object which models the LAV or GAV representation of the map. The parsing is done by applying the translation $T$ from Table 1. A MapView is pair of an ontology and a set of peer maps. The source descriptions which are described using the REL statements are parsed into the system as a SourceView object. A SourceView is a pair of an ontology and a set of source descriptions. Recall from Section 3, a source description contains a map that describes the assertion a source is making about some class or property and the URL of the source.

**MapKB:** This module maintains a collection of MapView objects and a collection of SourceView objects each indexed by their “source” ontologies. The source ontology of a map is the left hand side (LHS) ontology of a GAV map or the right hand side (RHS) ontology of a LAV map. This is the ontology whose classes or properties are being translated using the map. We refer to the other ontology (i.e. RHS of a GAV map or the LHS of a LAV map) as the target ontology. This is the ontology to which the source ontology’s terms are getting translated to. We use the indexing scheme described above to optimize the source selection algorithm.
**QueryHandler:** This module acts as the driver of the system during its query phase. It is responsible for parsing SPARQL queries into the system, issuing commands to the Reformulator to perform the source selection and loading the sources selected by the Reformulator into the AnsweringEngine. In addition to the sources this module loads the ontologies that are used in the reformulation and all the relevant maps. Then it issues the original query to the AnsweringEngine. Note: by loading only the used ontologies we provide a system that will scale well in terms of reasoning when we have potentially large number of ontologies.

**Reformulator:** This module implements our source selection algorithm. Our indexing scheme described above allows us to filter out the maps that are not useful in a given expansion. We observe that while expanding a goal node we only need to look at the maps that have the same source ontology as the goal node’s predicate. So we reduce the number of maps that are considered at any point during the reformulation. The output of this module is set of selected sources which may contribute to an answer.

**AnsweringEngine:** This module wraps the KAON2 reasoner and provides convenience methods for loading multiple ontologies and formatting returned answers. We note that since the OBII source selection algorithm is complete, OWLII is a sublanguage of OWL DL, and KAON2 is a complete reasoner for OWL DL, our system is complete for conjunctive queries over OWLII Semantic Web Space.

## 5 Evaluation

In this section we describe an initial set of experiments that examine the performance of our OBII system. In conducting these experiments we faced two obstacles. First, it is difficult to determine a realistic work load for this type of system. To address this issue we have generated several types of synthetic workloads to represent a wide variety of situations. Second, as mentioned in the related work section it is difficult to find systems that are comparable to us in their approach to solving the DQA problem in the Semantic Web. To address this issue we have developed a baseline system that loads all the ontologies and data sources and reasons over the complete knowledge base. This represents a single repository approach to the Semantic Web, which is essentially similar in spirit with the cache-based approach of search engines. We have chosen this as the baseline because there is considerable popularity of this type of architecture in the present Semantic Web [9]. However, in a dynamic data setting, this requires that all data be reloaded before the queries are processed. We have also developed a system that implements our source selection adaptation of PDMS without our Semantic Web based enhancements described in Section 4.1.

In what follows we first describe the workload generator and its parameters, then the workload configurations we chose to evaluate, the metrics we decided to collect and finally discuss some of our findings. All of our experiments were run on a Sun Fire X2100 Server with 8GB of main memory.

We implemented a workload generator that allows us to control several characteristics of our dataset. In generating the synthetic domain ontologies we decided to have on
the average 20 classes and 20 properties (influenced by the dominance of small ontologies in the current Semantic Web). The class and property taxonomy have an average branching factor of 4 and an average depth of 3. In generating the OWLII map ontologies we chose to have an even distribution of various OWLII axioms and chose to map about 30% of the classes and 30% of the properties of a given domain ontology. The resulting GAV and LAV maps contain an average of 5 conjuncts with some maps containing up to 11 conjuncts. The average data source has 75 triples and uses 30% of the classes and 30% of the properties of the domain ontology that it commits to. We generate 200 random queries with 1 to 3 conjuncts (75% are properties as opposed to a class) and containing a mix of distinguished and non-distinguished variables.

In choosing the configurations for our experiments we decided to vary two parameters: the number of data sources that commits to an ontology and the maximum number of maps required to translate from any source ontology to any target ontology. This number is referred to as the diameter by Halevy et al. [13]. We adopt the term in our discussion. We conducted two sets of experiments to evaluate the systems. In the first experiment (herein after referred to as experiment 1) we have varied the number of data sources that commits to a given ontology. In the second experiment (herein after referred to as experiment 2) we have varied the diameter. In both the experiments we kept the number of ontologies to 20. We denote an experiment configuration as follows: (nO-nD-nS) where nO is number of ontologies, n-D is the diameter and nS is the number of sources that commit to an ontology.

In our experiments the two main metrics we collected and examined are response time and the percentage of complete responses to queries. The response time is the time it takes from the issue of a query to the delivery of its result. We should note here that because the baseline system has a very different architecture, its response time is calculated differently. For the baseline system we add the load time (i.e. the time to load semantic web space) and the reasoning time to get the answers. The load time is added because as we are considering a dynamic environment, we should always work on fresh data, therefore each query results in a new knowledge base. For the other two systems, load time is calculated as the time to load the domain ontologies, the map ontologies that have been used in the source selection and the selected data sources. The response time for these two systems then is a sum of load time, source selection time and the reasoning time. In determining the completeness of queries, we consider the baseline system’s answers to be the reference set. This is reasonable because it has all the data available to it and uses the sound and complete KAON2 reasoner. In computing this metric we only consider queries that entail at least one answer.

The first observation from our experiments is that OBII is complete, where as the basic PDMS drops in completeness as we add data sources or increase the diameter. This is evident from Figure 2(a) which shows the average completeness over the set of queries for experiment 1. The second observation is that this completeness comes at a price. In figures 2(b) and 3(a) where we show the average response time for each system it is clear that OBII is twice as slow as the basic PDMS. However, OBII is 10 times faster than the baseline system, which is essentially a naive approach to achieving completeness.
This time penalty versus the PDMS is essentially unavoidable. In order to get complete answers, OBII loads more sources than PDMS, and the cost of loading these sources generally dominates its response time. This is evident from the figure 3(b). However, in higher diameter this dominance is reduced as OBII works with deeper rule goal trees. We note however even in the highest diameter experiment the response time for OBII is little over 2.5 seconds (as opposed to PDMS’s 1 second). Furthermore, PDMS is only 60% complete in that scenario.

6 Related Work

In recent years a few DQA systems have emerged that work on Semantic Web data. DRAGO [18] is theoretically grounded on the Distributed Description Logics framework (DDL). DDL extends standard Description Logics with means for expressing semantic mappings. Reasoning with multiple semantically related ontologies is accomplished by the use of semantic mappings to combine the inferences of local reasoning of each ontology. Although their original work focused only on TBoxes they have recently extended this work to accommodate ABoxes to perform instance retrieval queries [17].
Their work is different from ours in that they consider the map processing (translating query to source) as part of the reasoning process. Therefore they have to work on a much larger knowledge base as they have to consider all the maps available to the system.

The Piazza system [12] that uses the PDMS algorithm focuses more on integrating XML documents. The SWIM middleware [5] presents a similar work. The treatment of OWL is limited in these papers. The Piazza paper describes it as a fairly difficult problem and SWIM only works on RDF(S) maps.

Haase and Motik [10] have described a mapping system for OWL and proposed a query answering algorithm. They identify a mapping language that is similar to ours. However, as their language adds rules to OWL, it is undecidable and as such they need to introduce restrictions to achieve decidability. Our language, on the other hand, is a sublanguage of a decidable language. Furthermore, similar to the DRAGO approach, Haase and Motik do not rely on an explicit reformulation step and process all the maps for a query reformulation.

Peer-to-peer systems like Bibster [3] and SomeWhere [1] have shown promises in providing query answering solutions for the Semantic Web. However, a peer-to-peer system needs special software installed at every server. Our system on the other hand makes use of the existing infrastructure of the Web, as described in Observation 1 (see Section 1).

There has been some work done in selecting the appropriate RDF source for a query. One approach is to develop an index of sources [19] and develop a query answering algorithm that exploits these indexes. However, their source description language is limited to RDF whereas our OWLII is a much richer language for source description.

7 Conclusion and Future Work

In this paper we have introduced a source selection problem for the Semantic Web. We have defined OWLII, a subset of OWL that is compatible with the GAV and LAV formalisms and which is more expressive than DHL. We have adapted a query reformulation algorithm to solve our source selection problem and have shown how our solution is complete for conjunctive queries, OWLII maps and simple ontologies. As our experiments demonstrate our system is 10 times faster than a naive approach to complete reasoning, and only twice as slow as the incomplete information integration algorithm that it is based on.

This work opens up some interesting avenues for further research. First, we have assumed that the mapped ontologies only have simple taxonomic axioms. We have not considered the more advanced axioms that are available in OWL. One way to address this is to view the axioms as self-referential maps. Second, we want to investigate ways that will determine the best path to a translation. This may not always be the shortest path. Sometimes due to a translation that loses information, we may choose to follow a path that results in the least loss of information as opposed to the least number of translations. Third, we intend to explore methods of automatically generating high quality REL statements. Finally, we have observed that our rule-goal trees get very big as we
increase the diameter of the system. We intend to explore optimizations that remove redundancy from these trees.

8 Acknowledgment

This work was funded by Department of Energy via a Tech-X Corporation subcontract. We thank Ameet Chitnis and Fabiana Prabhakar for comments on this paper.

References