OPTIMISING ONTOLOGY INTEGRATION THROUGH INTERMEDIATE ONTOLOGIES

1V. RAJESWARI, 2Dr. DHARMISHTAN K. VARUGHESE

1Faculty, IT Dept., 2Faculty, ECE Dept., Karpagam College of Engg., Coimbatore, INDIA
E-mail: 1 rajeswari.vp21@gmail.com , 2dr.dharmishtan@gmail.com

ABSTRACT

The semantic web is increasingly being seen as a solution to manage knowledge content among heterogeneous and distributed information on the internet. Evolution of the semantic web is linked to a great extent to the evolution of various domain ontologies. It is necessary to formally define the mapping between ontologies to enable interoperability between applications in heterogeneous distributed information systems. The authors, in this paper illustrate how the fundamental problem of mapping between the global ontology and the local ontologies can be addressed, primarily through a newly developed WeGO algorithm. A mapping system for OWL-DL ontologies, where mappings are expressed as correspondences between conjunctive queries over ontologies, forms the core of this research work. The algorithm finds the semantically equivalent terms in local ontologies and uses them to build an intermediate ontology. The intermediate ontologies form the building block for a global ontology that will encompass the salient elements of the various local ontologies. It is further shown how the mapping system proves effective for the task of ontology integration through illustrative queries. Experimental data show that the query results obtained from the local ontology and global ontology match the results obtained from the intermediate ontology.

Keywords: Data Integration, RDF, OWL, Semantic Web, Heterogeneous Data, World Wide Web, Local Ontology, Global Ontology, Intermediate Ontology, Mapping, Merging

1. INTRODUCTION

1.1 Research Issue

In this era of coming together of “Knowledge Communities” across the globe, Semantic web is increasingly seen as a solution to manage content and knowledge among distributed information sources. Semantic web, to a large extent, is employing ontology for ensuring relevant information retrieval from diverse information sources. The main issue which is addressed by ontology is the semantic interoperability [1]. For achieving this, it is necessary to formally define the mapping between ontologies to enable interoperability between applications in distributed information systems [2]. In this paper, we define a mapping system for OWL-DL ontologies, where mappings are expressed as correspondences between conjunctive queries over ontologies [3]. We further show how the mapping system can be applied for the task of ontology integration and present a query system.

Using the graph representation for ontologies and schemas we proceed to calculate the weights for each node of the graph using the lexical similarity. The path traversed to reach a node is taken into consideration for matching different graphs which represent ontology or schema. Since the algorithm is very fast it can be used as a quick and primary method to do initial matching of a large dataset and then proceed to the exact match with other algorithms.

Semantic mapping for ontology development across different user communities has been an important research area. Theoretically it is possible to develop a global ontology carrying the same meaning for all distributed applications. But practical situations show otherwise [4], since different communities develop their own ontologies independently according to their interpretation of things. This necessitates a mapping method for enabling applications to exchange data and provide interoperability. Most of the mapping methods are based on standards of linguistic and structural characteristic similarity

1.2 Related work

Ontology based applications should harmonize their own ontologies to achieve semantic integration. This problem is known as ontology
alignment (matching) problem. The aim here is to find matches and relations [5] between concepts between different ontologies. Many mapping algorithms are recommended for ontology mapping [3]. Especially in [6], ontology mapping problem is indicated with comprehensible current solution approaches and a correct definition. As a general thing, today’s techniques make use of some research areas such as Bayes decision theory [3], information retrieval [5] and description logics [4].

Some of the popular algorithms that are in use for ontology mapping have approaches that are computation intensive and aim at larger ontologies. Anchor-flood algorithm [7] consists of two parts. The first one is ontology schema matching Anchor-flood algorithm ranging a set of ontology concepts and properties. Second one is instance matching. The weak point is the fact that this system ignores some distantly placed aligned pairs in ontology alignment system. In instance matching, it has still, rooms to work in structural transformation.

Another system AROMA [8] has three phases: (1) A preprocess phase that represents each title with a set of expressions like classes and properties, (2) The second phase consists of the occurrence of rules among labels, (3) A post-process phase that aims to increase the result mapping correctness and to elect unnecessary matches. Since AROMA returns not only equivalence correspondences but also subsumption correspondences, its precision value is negatively influenced. In [3], Choi et al. divided ontology matching approaches to three groups. The first group talks about mapping local and global ontologies. In this way, finding the relationship among local ontologies and a global ontology is an easy task because of a shared vocabulary that relates all the concepts in the local ontologies to the same concept in the global ontology. However, mapping local ontologies to each other becomes a hard task [9]. On the other hand, the second group talks about mapping local ontologies to each other. It maps similar concepts of source ontology to semantically related concepts in target ontology. This mapping technique is more appropriate for scaling up to the Web. The last group is based on merging ontologies to build a single coherent merged ontology [9]. It should be noted that the present work discusses only the first group in this paper.

2. ARCHITECTURE OF THE FRAME WORK

The standard information system architecture framework that handles user queries and interacts with the deep web is of the form shown in Fig. 1. The present work operates on the semantic layer of this architecture.

![Fig 1: Standard Information System Framework](imageURL)

The framework is based on the Integrated View (IV) and a set of wrappers. In this framework, IV follows a Local as View (LAV) approach to represent the mapping between the concepts in the source ontologies and the integrated view.

The Transformation Processor (TP) transforms the data from the data source model to the materialized data model. In the implementation, we consider the materialized data being represented as an ontology model [9]. During the maintenance of the materialized view, according to the updated occurring in the data sources, the Incremental Maintenance Processor (IMP) will determine which data in the materialized view are going to be updated. After the IMP receives the integrated data from IV, it will compare to decide which parts need to be updated [11].

The two modules TP and IMP in the dashed box in the figure form the Maintenance module for the Materialized View (MMV). The task of the Query Processor QP in this architecture is to determine if the query could be answered from the materialized
3. GLOBAL AND LOCAL ONTOLOGY

Ontology helps in semantically connecting references in relation to the context they occur. Though it is possible to ideally visualise a Global Ontology that encompasses everything, we believe that creation of an Intermediate Ontology simplifies the mapping and merging. Local ontologies enable local groups to build, maintain and use their own interpretation for Ontologies.

3.1 Global Ontology

We denote with $\mathcal{A}_G$ the alphabet of terms of the global ontology, and we assume that the global ontology of an Ontology Information System (OIS) is expressed as a theory $\mathcal{G}$ in some logic $\mathcal{L}_G$ [2].

3.2 Local Ontologies

We assume to have a set $\mathcal{S}$ of ‘$n$’ local ontologies $\mathcal{S}_1 \ldots \mathcal{S}_n$. We denote with $\mathcal{A}_{\mathcal{S}_i}$ the alphabet of terms of the local ontology $\mathcal{S}_i$. We also denote with $\mathcal{A}_S$ the Union of all $\mathcal{A}_{\mathcal{S}_i}$’s. We assume that the various $\mathcal{A}_{\mathcal{S}_i}$’s are mutually disjoint, and each one is disjoint from the alphabet $\mathcal{A}_G$. We assume that each local ontology is expressed as a theory $\mathcal{S}_i$ in some logic $\mathcal{L}_{\mathcal{S}_i}$, and we use $\mathcal{S}_{\mathcal{S}_i}$ denote the collection of theories $\mathcal{S}_1 \ldots \mathcal{S}_n$.

3.3 Mapping

The mapping $\mathcal{M}_{\mathcal{G}, \mathcal{S}}$ is the heart of the OIS, in that it specifies how the concepts in the global ontology and the local ontologies map to each other.

3.4 Semantics

Intuitively, in specifying the semantics of an OIS, we have to start with a model of the local ontologies. The crucial point is to specify which are the models of the global ontology that need to be considered [10]. Thus, for assigning semantics to an OIS $\mathcal{O} = \langle \mathcal{G}, \mathcal{S}, \mathcal{M}_{\mathcal{G}, \mathcal{S}} \rangle$, we start by considering a local model $\mathcal{D}$ for $\mathcal{O}$, i.e., an interpretation that is a model for all the theories of $\mathcal{S}$. We call global interpretation for $\mathcal{O}$ any interpretation for $\mathcal{G}$. A global interpretation $\mathcal{I}$ for $\mathcal{O}$ is said to be a global model for $\mathcal{S}$ with respect to $\mathcal{D}$ if:

- $\mathcal{I}$ is a model of $\mathcal{G}$, and
- $\mathcal{I}$ satisfies the mapping $\mathcal{M}_{\mathcal{G}, \mathcal{S}}$ wrt $\mathcal{D}$.

Following are the research done in data integration [10]. The two basic approaches for defining this mapping are as follows:

In the global-centric approach the concepts of the global ontology $\mathcal{G}$ are mapped into queries over the local ontologies in $\mathcal{S}$ where as in the local-centric approach the concepts of the local ontologies in $\mathcal{S}$ are mapped into queries over the global ontology $\mathcal{G}$.

4. CASE 1: LOCAL ONTOLOGIES

Two local ontologies $\mathcal{MGI}$ and $\mathcal{MCK}$ are derived from the corresponding database of the respective websites www.medguideindia.com and www.medelik.com. These websites provide information on drugs and medicines and information on diseases and treatment.

4.1 Local Ontology $\mathcal{MGI}$

The Drugs, Immunization and the Health Insurance are the three major categories of the local ontology $\mathcal{MGI}$, shown in Fig. 2.
4.1.2 Applying relational algebra

The ‘Drugs’ is a subclass of MGI. The ‘Project’ operation is applied to the Drugs, which is the union of Brand, Manufacturer and Generic.

Drugs = Brand\_brand,Category,Constituents\_per\_unit,Manufacturer\_brand,Package\_per\_unit,Price\_per\_unit,PriceInRS\_brand,Sno,Type\_brand,Unit\_brand\( \cup \)Indication\_Contradication\_Precaution\_Sideeffects,MatchedBrandsWith\_Combination\_Of\_Generics,MatchedBrands\_with\_Single\_Generic,Sno\_generic\( \cup \)Address,BrandList(),Email,Fax,Manufacturer\_Name,Phone\_No,S\_No,Sub\_Division,URL\( \cup \)Drugs

4.1.3 Applying the axiom property II to MGI

Constructor : hasClass and hasValue
DL Syntax : \exists P.C and \exists P.\{X\}
Pattern : \exists Local Ontology. hasClass. \{f_1, f_2, f_3… f_n\} where fi, f2, f3…fn indicates Fields
The utility of the knowledge base as interlinking results based on Ontologies conceptual structure. very much reduced. In the context of classic instances used for the efficient query retrieval is answer the query. By our approach the number of RDF links to discover data that might be relevant to we propose a query execution with the traversal of SPARQL, the knowledge base can be used to relate search terms to entities and to improve search results based on Ontologies conceptual structure. The utility of the knowledge base as interlinking hub for the Web of Data is demonstrated by the models. It can be done equally well on the OWL local ontologies through working on their RDF links [8].

5 \textbf{CASE 2: GLOBAL ONTOLOGY}

The global ontology formed using the instances of MGI and MCK, Global Ontology \(XY\) is formed as the UNION of the Local Ontologies. The common element is retrieved by using the notation \(X \cup Y\).

\textbf{Global ontology} = \textit{MGI} \cup \textit{MCK}

The global ontology is built by unioning the local ontologies through working on their RDF models. It can be done equally well on the OWL model also.
5.1 RDF Dataset

An RDF dataset [13] is a set \( D = \{ G_0, u_1, G_1, \ldots, u_n, G_n \} \) where \( G_0, \ldots, G_n \) are RDF graphs, \( u_1, \ldots, u_n \) are IRIs, and \( n \geq 0 \). In the dataset, \( G_0 \) is the default graph, and the pairs \( u_i, G_i \) are named graphs, with \( u_i \) the name of \( G_i \). Every dataset \( D \) is equipped with a function \( d_D \) such that \( d_D(u) = G \) if \( u \in D \) and \( d_D(u) = \emptyset \) otherwise. Additionally, \( \text{name}(D) \) stands for the set of IRIs that are names of graphs in \( D \), and \( \text{term}(D) \) and \( \text{blank}(D) \) stand for the set of terms and blank nodes appearing in the graphs of \( D \), respectively. For the sake of simplicity, we assume that the graphs in a dataset have disjoint sets of blank nodes, i.e. for \( i \neq j \), \( \text{blank}(G_i) \cap \text{blank}(G_j) = \emptyset \).

5.2 Mapping

A mapping \( \mu \) from \( V \) to \( T \) is a partial function \( \mu : V \rightarrow T \). The domain of \( \mu \), \( \text{dom}(\mu) \), is the subset of \( V \) where \( \mu \) is defined. The empty mapping \( \emptyset \) is a mapping such that \( \text{dom}(\emptyset) = \emptyset \) (i.e. \( \mu = \emptyset \)).

The RDF mapping API allows the data for easy re-use by mapping bundles to RDF types and fields to RDF predicates. This abstract mapping can then be used to publish the content contained in these bundles, enabling the data to be serialised into a number of different formats, such as RDF, RDF/XML, or populate a SPARQL endpoint.

```
PREFIX foaf: <http://xmlns.com/foaf/0.1/>
PREFIX rdf: <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
FILTER (?Brand_brand = ?Brand_Name) ORDER BY ASC(?Brand_Name) LIMIT 50
```

Figure 5: SPARQL query for Global Ontology

6 CASE 3: INTERMEDIATE ONTOLOGY

The intermediate ontology is formed through the application of the WeGO algorithm which checks the semantic equivalence of nodes in MGI, MCK and the global ontology.

6.1 The WeGO Algorithm

The Weighted Global Ontology Algorithm uses a simple method of matching entities by comparing the location on the ontology tree. Procedure ‘find’ first checks whether exactly two nodes are present with the same name in both graphs by using procedure ‘compare’, and returns if successful. If not, the flag is set to search based upon the synsets of nodes values. Procedure ‘compare’ uses Breadth First Search (BFS) methodology. It works by enqueuing the root node. The node is de-queued for examination, if the element sought is found in this node, then the element is passed to the procedure ‘searchGraph’ and checked whether the node value from ‘searchGraph’ is equal to the value of the current element. If the values are equal, the search is returned along with their weights, otherwise enqueue any successors (the direct child nodes) that have not yet been discovered. If the queue is empty, every node on the graph has been examined – quit the search and return "null". Repeat until the queue is not empty. The procedure ‘searchGraph’ uses BFS to navigate through all its nodes. The flag ‘exactMatch’ decides whether to use the synsets of the corresponding node value or only the node value. Procedures ‘isSimilarEntities’ and ‘getSynsets’ retrieve the synsets of a particular node value from the Wordnet database which is used upon by the calling procedure ‘searchGraph’. On successful matching of node values, procedure ‘find’ returns the node values along with the corresponding weights from both the graphs in the defined ‘word’ structure.
procedure find(G, G', word):
    valueFound = compare(G,G',word,true)
    if(valueFound!=null)
        return valueFound
    valueFound = compare(G,G',word,false)
    if(valueFound!=null)
        return valueFound
    return null

procedure compare(G, G', word, exactmatch):
    depth ← 0
    create a queue Queue_G
    enqueue root_G onto Queue_G
    mark root_G
    while Queue_G is not empty:
        t ← Queue_G.dequeue()
        t.weight_G ← depth
        if (t == word.value)
            foundWord ← searchGraph(G',t,exactMatch)
            if (foundWord.value == t):
                t.weight_G' ← foundWord.weight_G'
                return t
        for all edges edge_G in G.adjacentEdges(t) do
            nextVertex_G ← G.adjacentVertex(t,edge_G)
            if nextVertex_G is not marked:
                mark nextVertex_G
                enqueue nextVertex_G onto Queue_G
                depth ← depth + 1
        return null

procedure searchGraph(G', wordToBeFound, exactMatch):
    depth ← 0
    create a queue Queue_G'
    enqueue wordToBeFound onto Queue_G'
    mark wordToBeFound
    while Queue_G' is not empty:
        currentWord ← Queue_G'.dequeue()
        currentWord.weight_G' ← depth
        if (exactMatch):
            if (currentWord.value == wordToBeFound.value):
                return currentWord
        else
            if(isSimilarEntities(wordToBeFound.value, currentWord.value)):
                return currentWord
        for all edges edge_G' in G'.adjacentEdges(currentWord) do
            nextVertex_G' ← G'.adjacentVertex(currentWord,edge_G')
            if nextVertex_G' is not marked:
                mark nextVertex_G'
                enqueue nextVertex_G' onto Queue_G'
                depth ← depth + 1
                return null

procedure isSimilarEntities(baseEntity, candidateEntity):
    baseSynsets ← getSynsets(baseEntity)
    if (baseSynsets.contains(candidateEntity)):
        return true
    return false

procedure getSynsets(entity):
    WordNetDatabase database ← WordNetDatabase.getFileInstance()
    Synset[] synsets ← database.getSynsets(entity)
    while i less than synsets.length():
        String[] wordForms ← synsets[i].getWordForms()
        while (j less than wordForms.length()):
            synsetData.add(wordForms[j])
        return synsetData

Figure 7: WeGO Algorithm for Ontology

The Fig. 9 and Fig. 10 depict the nodes of local ontologies MGI and MCK. The java application which executes the WeGO algorithm, automatically detects the nodes and their levels in the graph. The ‘Value’ and the ‘Level’ of the tree are indicated as a graph (MGI) in Fig. 8 below.

Figure 8: Ontology Tree of MGI

The node which is to be tested is given as input in the graph. The application give the semantic equivalent in the graph (MCK) and its relevant structure. From the axiom of DAML+OIL in conjunction with the algorithm, the application produced the following semantic classes that were found to match between MGI and MCK.

- Manufacturer_brand ≡ Manufacturer
- Type_brand ≡ Type_of_Brand
6.2 Evaluation Terms Used

6.2.1 Evaluation semantics
We define eval (D(G), graph pattern) as the evaluation of a graph pattern with respect to a dataset D having active graph

- G: The active graph is initially the default graph. The active graph is used to match the pattern unless otherwise stated.
- D: a dataset
- D(G): D a dataset with active graph G (the one patterns match against)
- D[i]: The graph with IRI i in dataset D
- D[DFT]: the default graph of D

6.2.2 Filter
Let Ω be a multiset of mappings and ‘expr’ be an expression. We define:

- Filter (expr, Ω) = { μ | μ in Ω and expr(μ) is an expression that has a boolean effective value of true }
- card [Filter(expr, Ω)](μ) = card[Ω](μ)

6.2.3 Join
Let Ω1 and Ω2 be multisets of mappings. We define:

- Join(Ω1, Ω2) = { merge(μ1, μ2) | μ1 in Ω1 and μ2 in Ω2, and μ1 and μ2 are compatible } 
- card[Join(Ω1, Ω2)](μ) = sum over μ in (Ω1 set-union Ω2), card[Ω1](μ1)*card[Ω2](μ2)

6.2.4 Evaluation of Join (P1, P2, F)
- eval(D(G),Join(P1,P2)) = Join(eval(D(G), P1), eval(D(G), P2))
- eval(D(G),Join(P1,P2, F) = Filter(F, Join( eval(D(G), P1), eval(D(G), P2) )))

6.2.5 Solution modifiers
The notations used in the SPARQL after parsing are as follows.

- DISTINCT
- PROJECT
- ORDER BY
- LIMIT/OFFSET

The intermediate ontology is tested with the three different cases.

- Brand: {Brand _brand≡ Brand _Name} 
- Manufacturer:{Manufacturer_brand≡ Manufacturer} 
- Type:{Type_brand ≡ Type_of_Brand}

For the above semantic classes 1192 sets of intermediate record set are tested. The equivalent sets if records in the second ontology are tested against the Local Ontology 1. The total number of result set obtained in case 1 and case II are equivalent to the case III

By using the Intermediate Ontology the results obtained are found to be similar to the Case 1 – Local Ontologies and Case 2 – Global Ontology.

6.3 Comparative Query Analysis of Ontologies

The local ontologies MGI and MCK are taken for analysis. For deriving the intermediate ontology, the common elements are retrieved and all the instances of the common elements are included. If X and Y are two local ontologies, then common elements are retrieved by using the notation X ∏ Y.

The global ontology was formed by the union of local ontologies MGI and MCK. Intermediate ontology is formed using the instances of MGI and MCI. If X and Y are two local ontologies, then the global ontology XY is formed as indicated by the notation X U Y.

- No. of instances in local ontology     - 32 
- No. of instances in global ontology    - 28 
- No. of instance in intermediate ontology - 08 

The total number of elements tested with the Brand is 211, with Manufacturer is 111 and with Type is 39.

7. CONCLUSION

A web with better defined semantic languages, with an increased expressivity and a wide area of covered domains, used everywhere in the simple possible way, in different corporations by non-expert users, will be the focus of future web applications. The authors believe the present work will contribute to such developments. Ontology mapping is concerned with reusing existing ontologies, expanding and combining them by some means and enabling a larger pool of information and knowledge in different domains [14] to be integrated to support new communication and use.
The work discussed in this paper shows a simple approach that forms the basic building block in forming a comprehensive ontology from such local ontologies. Ontology evolving, likewise, is concerned with maintaining existing ontologies and extending them as appropriate when new information or knowledge is acquired. This can be a starting point in integrating smaller intermediate ontologies to arrive at a truly global ontology through iterative methods that can evolve in future research work.

REFERENCES


Table 1: Comparison of Records,Instances and Intermediate Elements
(Total Number of Record Sets Tested: 1192)

<table>
<thead>
<tr>
<th>Class</th>
<th>Semantic Classes</th>
<th>Total No. of Records Retrieved</th>
<th>Total No. of Instances</th>
<th>Total No.of Intermediate elements tested under each category</th>
</tr>
</thead>
<tbody>
<tr>
<td></td>
<td></td>
<td>Local</td>
<td>Global</td>
<td>Intermediate</td>
</tr>
<tr>
<td>Manufac</td>
<td>Manufacturer_brand ≡ Manufacturer</td>
<td>62</td>
<td>62</td>
<td>62</td>
</tr>
<tr>
<td>Brand</td>
<td>Brand_brand ≡ Brand_Name</td>
<td>26</td>
<td>26</td>
<td>26</td>
</tr>
<tr>
<td>Type</td>
<td>Type_brand ≡ Type_of_Brand</td>
<td>99</td>
<td>99</td>
<td>99</td>
</tr>
</tbody>
</table>
Figure 9: Protégé Model of MGI

Figure 10: Protégé Model of MCK
Figure 11: Comparison - No. of Instances of Local, Global and Intermediate Ontology

Figure 12: Comparison - Query results from Local, Global and Intermediate Ontology