An Ontological Query Model to Support Question Answering in the Deep Web[1]

Christan GRANT [a,2], Clint P. GEORGE [a], Jeff T. DEPREE [a], Peter J. DOBBINS [a] and Joseph N. WILSON [a]

[a] *Database Research Center, Computer and Information Science and Engineering Department, University of Florida, Gainesville, FL, USA*

Abstract. The *deep web* holds millions of data sources that are available for use within question answering systems. But access to the information they provide is difficult because of ad-hoc search methods, varied structures, and restrictions imposed by terms of use. Efficient methods to discover and obtain these data are a key to developing more powerful question answering systems. In order to support discovery in this setting, we propose an ontological model to act as a *lingua franca* to translate between natural language imperative or interrogative clauses and deep web page interaction specifications. As a corollary, we present an ontology matching algorithm to support the translation of clauses to answers.

Keywords. Deep Web, NLP, Question Answering, Ontology Matching

Introduction

The *deep web* consists of all the on-line information that is not subject to direct search by traditional search engines. It is estimated to contain two to three orders of magnitude of structured data beyond the information that is currently accessible via traditional search engine techniques [6]. In addition to containing more information than the shallow web, the deep web is more dynamic and is not indexed. Methodologies are now being investigated to improve the ability of search and discovery engines to locate relevant deep web information [2,1,7,11].

Some early deep web resource discovery systems try to use sample queries to extract information from forms and index the resulting pages [5]. They then use entity extraction techniques to discover the type of information provided. However, sampling web forms can overload web servers and violate their terms of use agreements. Other systems create *wrappers* around the deep web form, that is, programs that treat web forms as functions with inputs and outputs. For these wrappers to support an intelligent deep web discovery engine, they must be accurate and robust.

Estimates from early work suggest that it takes users an average of 30 minutes to create a traditional wrapper for a deep website [8]. Consider a specific vertical market,

[1]The Morpheus Project is funded by NSF Grant 0712799

[2]Corresponding Author: Christan Grant, CSE E457-8, PO Box 116120, Gainesville, Florida, 32611-6120; Email: cgrant@cise.ufl.edu.

such as travel or aftermarket auto parts. Relevant deep web sources can be hosted on up to 10,000 different web servers, thus wrapping all of them would take about 5,000 hours, or 2.5 person years. It is clear that traditional deep web wrapping techniques cannot scale to solve such problems.

To address the problem of exploiting information in the deep web, we are developing a deep web question answering system. We use sample user interactions to build a large body of answerable questions. When presented with a question, a user searches the web, querying deep web sources to find an answer. We record the search process associated with answering each question. Questions are represented using an ontological model and the search process is formalized as a *query resolution mechanism* (QRM).

We choose to represent queries using ontologies because they provide a proven method of knowledge transfer [3]. In particular, we employ the popular OWL markup language.[3] This choice allows us to exploit work in ontology alignment to compare new queries to example queries for which QRMs exist and also to rank those QRMs by their relevance.

This paper includes the following:

- It presents a model for supporting question answering with deep web page interactions.
- It provides an ontological model that acts as an intermediate language for natural language queries and their associated deep web page interactions.
- It presents an ontological matching algorithm that can be used to identify and rank relevant sample queries and their associated QRMs.

1. Ontology Representation

To motivate our modeling problem, we consider the following question: "What is the cheapest five-star hotel in Zurich on October 4, 2010?" Natural language queries of this sort cannot be processed directly by our current system. Instead, we use a *semi-structured query* (SSQ) representation. Each SSQ is associated with a realm (such as travel, sporting events, recreation, etc.) and can contain four types of contextual clauses: *who, what, when,* and *where*, possibly having modifiers. Each context represents an input or an output of the query but not both. For our example, the inputs are *where* (Zurich) and *when* (4, October 2010), and the output is *what* (cheapest five-star hotel).

If an input or an output of an SSQ has a contextual class association, it is said to be *qualified*. Each input and output of the sample queries used to build our QRMs is qualified, thus, we refer to these as fully-qualified SSQs.

1.1. Deep Web Model

Answering a question using the deep web requires one to navigate through a sequence of one or more web pages and links. Many of the accesses involve clicks through web forms to resulting pages. We have developed a model that represents the various actions a user may carry out during this process. We record the user actions with a browser plugin called the *query resolution recorder* (QRR).

[3]OWL Web Ontology Language. http://www.w3.org/TR/owl-ref/

During the answer retrieval process, a user discovers an element of the query's result on a web page. Next, she highlights that element, typically a text string. This highlight allows the QRR to record the source of the query's answer. The QRR stores all information from the user query in our data store. When the user submits a web form she may associate each of the form's inputs with one of our stored context classes.

The discovery process can be broken down into three cases:

1. The user typed in a URL and highlighted part of or all the answer.
2. The user typed in a URL, clicked a link, and found part or all of the answer.
3. The user typed in a URL, filled out a form, and found part or all of the answer on a subsequent page.

Any discovery process consists of a sequence of these transactions. Additionally, data collected for inputs and outputs may be found during any stage of the complete discovery process. An output collected during one stage of an extraction may be the input to a subsequent stage in the extraction. The QRR builds the completed user collection process into a QRM. The following is a mathematical formulation of information collected for user web interactions.

A QRM is a 5-tuple $Z = \langle \Upsilon, \Omega, P, M, R, A \rangle$ such that:

1. Υ is the sequence of input classes $\langle \Upsilon_1, \Upsilon_2, \Upsilon_3, \Upsilon_4 \rangle$ where Υ_1 represents the *who* context of the associated query, Υ_2 represents the *what* context, Υ_3 represents the *when* context, and Υ_4 represents the *where* context.
2. Ω is the sequence of output classes $\langle \Omega_1, \Omega_2, \Omega_3, \Omega_4 \rangle$ where Ω_1 represents the *who* context of the associated query, Ω_2 represents the *what* context, Ω_3 represents the *when* context, and Ω_4 represents the *where* context.
3. P is an ordered list of web pages $P = \langle P_1, P_2, ..., P_n \rangle$ where $P_h = \langle U_h, \langle I_{h_1}, I_{h_2}, ..., I_{h_u} \rangle, \langle O_{h_1}, O_{h_2}, ...O_{h_v} \rangle \rangle$ for $1 \leq h \leq n$, $u = |I_h|$, $v = |O_h|$. P_h is a triple comprised of a URL U_h, a sequence of input arguments, and a sequence of output results.
   Let $I_\$ = \bigcup_{j=1}^{4} I_{\downarrow_j}$ be the set of the selector expressions for the inputs of a query and $O_\$ = \bigcup_{j=1}^{4} O_{\downarrow_j}$ be the set of selectors expressions for the outputs of a query. Let K be the set of all string constants.
   - URI $U_1 \in K$ is a string and
     for $1 < g \leq n$, URI $U_g \in K \cup \left(\bigcup_{p=1}^{g-1} \left(\bigcup_{h=1}^{|O_p|} O_{p_h} \right) \right)$ where $1 \leq h \leq |O_p|$,
   - for $1 < h < |I_1|$, input $I_{1_h} \in K \cup I_\$ \cup O_\$$ and
     for $1 < g <= n$, then for $1 < h < |I_g|$, input $I_{g_h} \in K \cup I_\$ \cup O_\$ \cup \left(\bigcup_{p=1}^{g-1} \left(\bigcup_{h=1}^{|O_p|} O_{p_h} \right) \right)$.
4. M is a map between Υ, K, Ω to page list inputs I_h.
5. R is an ontological realm.
6. A is the sequence of outputs from Υ, K, and Ω representing the answer.

1.2. Query Model

It is straight-forward to translate a fully-qualified SSQ, represented by Υ and Ω, into a class in an OWL ontology. Any ontology that characterizes SSQs must define object properties for each of the different contexts that qualify the inputs and outputs of the query.

A member of the Context class contains a reference to an individual belonging to a given Context, as well as zero or more modifiers that determine the desired qualities of the referenced individual. In most cases, the Context properties contained within an SSQ will reference subclasses of this base Context class. Here is the hotel Context from the motivating example given above.

```
<owl:Class rdf:ID="HotelContext">
   <rdfs:subClassOf rdf:resource="#ContextClass" />
   <owl:Restriction>
      <owl:onProperty rdf:resource="#ValueWithinContext" />
      <owl:maxCardinality
         rdf:datatype="&xsd;nonNegativeInteger">
         1</owl:maxCardinality>
      <owl:allValuesFrom rdf:resource="#Hotel" />
   </owl:Restriction>
   <owl:Restriction>
      <owl:onProperty rdf:resource="#Modifier" />
      <owl:hasValue rdf:resource="#Cheapest" />
   </owl:Restriction>
   <owl:Restriction>
      <owl:onProperty rdf:resource="#Modifier" />
      <owl:hasValue rdf:resource="#Five-Star" />
   </owl:Restriction>
</owl:Class>
```

The object property *hasRealm* maps each query to the subject matter to which it relates. The elements that can serve as the realm of a query can be individuals of any type.

For each SSQ, a class is constructed that defines a specific type of query which has input and output contexts that belong to a limited set of classes. Here is the SSQ class from the motivating example given above.

```
<owl:Class rdf:ID="SSQ-0">
   <hasRealm rdf:resource="#Travel"/>
   <owl:Restriction>
      <owl:onProperty rdf:resource="#Where_IN" />
      <owl:allValuesFrom rdf:resource="#CityContext" />
   </owl:Restriction>
   <owl:Restriction>
      <owl:onProperty rdf:resource="#When_IN" />
      <owl:allValuesFrom rdf:resource="#DateContext" />
   </owl:Restriction>
   <owl:Restriction>
      <owl:onProperty rdf:resource="#What_OUT" />
      <owl:allValuesFrom rdf:resource="#HotelContext" />
   </owl:Restriction>
```

```
</owl:Class>
```

This class restricts its properties in such a way that it will only be matched to other SSQs if they involve inputs and outputs from the same classes, that is, ones that make queries about hotels for a given city and date.

2. Ontology Matching

In order to match a new query with a QRM within our data store, we have defined similarity measures between the ontologies of the new query and the canidate QRMs. These measures capture the distance between the ontology classes, properties, and individuals at the conceptual level in addition to simply matching URIs. The similarity measures are implemented in two scenarios: *online execution* and *offline mode*. The *online execution* is time sensitive and is used primarily for relating newly defined QRMs to the QRMs in the store. During *offline execution*, the system clusters similar QRMs.

The similarity measure for the QRM ontologies is determined by the following components.

1. QueryClass: an OWL class and the root of the QRM ontology structure.
2. hasRealm: an OWL object property with its domain as the QRM Query class and its range as a realm class.
3. Contexts such as who, what, when, and where are OWL object properties, whose domains are the query classes and ranges are OWL classes.
4. QRMClass: an OWL class which is created from the components described above.

Based on this ontology structure and the execution scenarios, we have created two algorithms for finding the matches between QRM ontologies. The first algorithm is based on class divergence, which is motivated by the concept of multiple dispatch in CLOS [9] and Dylan programming [4] for generic function type matches. We use this algorithm for *online excution*, where we consider the realm class and context property ranges for determining QRM similarity. The second algorithm is based on the ontology structure and semantics [10]. This algorithm is used for *offline execution*, where it finds the similarity between the QRM ontology classes, properties, and individuals and merges the similar ontology entities. Subsections 2.1, 2.2, and 2.3 explain how we can make use of our algorithms for finding the similarities between the QRM Ontologies.

2.1. Class Divergence

We employ a measure of compatibility, which we call *class divergence*, between a source class and a target class using the *topological structure* of the context classes in the ontology. We write $S \prec T$ for the reflexive transitive closure of the superclass relation. Let $d(P, Q)$ represent the hop distance in the directed ontology inheritance graph from P to Q. The divergence between a source and target class ranges from zero (for identical classes) to one (for incompatible classes). Let S be the source class, T be the target class, and let C be a common ancestor class of S and T minimizing $d(S, C) + d(T, C)$. The class divergence between S and T, the unnormalized class dissimilarity, is defined as follows:

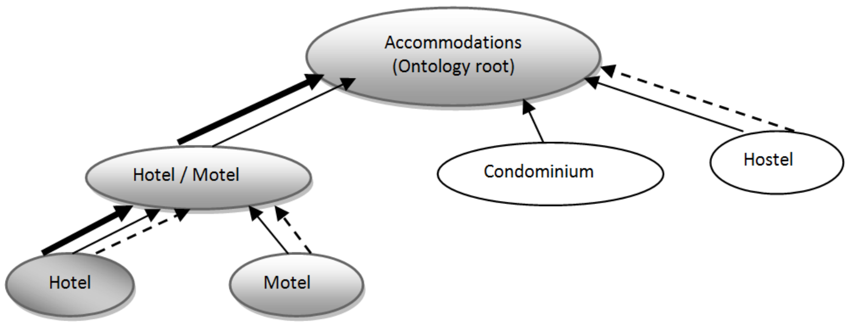

Figure 1. Graphical representation of an Ontology

$$d_{class}(S,T) = \begin{cases} 0 & if S.uri \equiv T.uri \\ d(S,T) & if S \prec T \\ 1 & if T \prec S \\ d(S,root) + d(S,C) + d(T,C) & otherwise \end{cases} \tag{1}$$

We normalize $d_{class}(S,T)$ by dividing by three times the height of the ontology tree.

Note, if $S \prec T$ and $S \not\prec Q$ then $d_{class}(S,T) < d_{class}(S,Q)$, that is, the divergence of a source class to a target ancestor class is smaller than the divergence of a source class to any class that is not an ancestor. This is an important property in determining the compatibility of classes for answering queries. If a QRM answers queries concerning an ancestor class, it is more relevant that a QRM that answers queries from any non-ancestral class.

Figure 1 illustrates the calculation of class divergence. This sample ontology is rooted at the class *Accommodations*. The arrows represent the inheritance relationship between the classes in the topology. Suppose we want to find the divergence $d_{class}(Hotel, Motel)$. Both are subclasses of the class *Hotel/Motel* (C). They share the path from *Hotel/Motel* to the $root$. So, the unnormalized divergence $d(Hotel, root)$ is two (shown by bold arrows), $d(Motel, Hotel/Motel)$ is one (dashed arrow), and $d(Hotel, Hotel/Motel)$ is one (dashed arrow). Thus, $d(Hotel, Motel)$ is four, which corresponds to a normalized value of $2/3$. Consider $d_{class}(Motel, Hostel)$. The classes $root$ and C in the definition of d_{class} are the same in this case so we add $d(Motel, Accommodations)$ (two), $d(Hostel, Accommodations)$ (one), and $d(Motel, Accommodations)$ (two), yielding a total unnormalized divergence of five, or a normalized divergence of $5/6$.

2.2. *QRM Ontology Matching*

We use class divergence to calculate the relevance between a source SSQ and a target SSQ associated with a QRM. Each SSQ will have four input classes, four output classes, and one realm. We refer to the sequence of these classes $\langle \Upsilon_1, ..., \Upsilon_4, \Omega_1, ..., \Omega_4, R\rangle$ as the signature of an SSQ. If an input or output class is unspecified in the query, it is associated with the class $bottom$, which has no subclasses or individuals.

The relevance of QRM Q with signature $\langle \Upsilon_{Q_1}, ..., \Upsilon_{Q_4}, \Omega_{Q_1}, ..., \Omega_{Q_4}, R_Q \rangle$ to SSQ S with signature $\langle \Upsilon_{S_1}, ..., \Upsilon_{S_4}, \Omega_{S_1}, ..., \Omega_{S_4}, R_S \rangle$ is

$$R(S,Q) = 1 - \pi_9 d_{class}(R_S, R_Q) + \sum_{k=1}^{4} \pi_k d_{class}(\Upsilon_{S_k}, \Upsilon_{Q_k}) + \sum_{j=1}^{4} \pi_{4+j} d_{class}(\Omega_{S_j}, \Omega_{Q_j}) \tag{2}$$

where the weights π_k of the property hasRealm and other context properties satisfies the conditions

$$\sum_{k=1}^{9} \pi_k = 1 \tag{3}$$

and for $1 \leq i \leq 9$

$$\pi_i > 0 \tag{4}$$

If an SSQ and a QRM have identical classes for all inputs, all outputs, and their realms then the QRM relevance value will be one. Otherwise, the relevance of the QRM will be in the range $[0, 1)$.

2.3. *Ontological Structural Similarity*

Employing user input brings about the likelihood that redundant ontological elements will be inserted into our context class ontology. The work described here is aimed at finding ontological similarities between our context classes and QRMs so that we can reduce the number of redundant and incomparable elements. The ontological structural measure is determined by examining the ontological structure and semantics of each QRM. Our analysis depends on the ontology entities such as classes, individuals, properties, superclasses, and subclasses. The structural similarity is obtained from the following dissimilarity measures.

(a) class dissimilarity: This measure $(dist_{class})$ is determined by these features of the source and target classes: class URIs, parent classes, and class properties.

(b) property dissimilarity: This measure $(dist_{property})$ is determined by these features of the source and target class properties: property URIs, property domains $(classes)$, and property ranges $(classes)$.

(c) individual dissimilarity: This measure $(dist_{individual})$ for the source and target individuals is determined by their URIs, properties, and class types.

2.3.1. *Class dissimilarity*

Let S be a source class and T be a target class. The set of all the parent classes of the class C is $parent(C)$ and the set of all the properties of the class C is $property(C)$. We define two levels of class dissimilarity.

Level 0: In this level, the distance is measured using the dissimilarity between the OWL entity URIs $(dist_{URI})$:

1. URI dissimilarity for source and target classes
   $f_1(S,T) = dist_{URI}(S.uri, T.uri)$
2. URI dissimilarity for all the class properties
   $f_2(S,T) = dist_{URI}(property_i.uri, property_j.uri)$,
   $\forall property_i \in property(S), \quad \forall property_j \in property(T)$
3. URI dissimilarity for all the parent classes
   $f_3(S,T) = dist_{URI}(class_i.uri, class_j.uri)$,
   $\forall class_i \in parent(S), \quad \forall class_j \in parent(T)$

Level 1: Here we calculate the URI dissimilarity in the same manner as in Level 0. It calculates class dissimilarity [Eq. (5)] and property dissimilarity [Eq. (8)] for each parent classes and the properties of the source and target classes. The two additional steps are:

1. Compute property dissimilarity for all of the class properties
   $f_4(S,T) = dist_{property}(property_i, property_j)$,
   $\forall property_i \in property(S), \quad \forall property_j \in property(T)$
2. Compute class dissimilarity for all of the super classes of the class
   $f_5(S,T) = dist_{class}(class_i, class_j)$,
   $\forall class_i \in parent(S), \quad \forall class_j \in parent(T)$

Finally, we combine all these distances into a combinatorial measure for the complete OWL class dissimilarity measure

$$dist_{class}(S,T) = \sum_{i=1}^{N} \alpha_i f_i(s,t) \tag{5}$$

where α_i is the weight for a feature i, N is the number of features, and satisfies the conditions

$$\sum_{i=1}^{N} \alpha_i = 1 \tag{6}$$

and

$$\alpha_i \geq 0 \tag{7}$$

2.3.2. Property dissimilarity

Let S_P be a source property and T_P be a target property. The set of all the domain classes of the OWL property P is $domain(P)$ and the set of all the range classes of the OWL property P is $range(P)$. We define two levels of property dissimilarity.

Level 0: In this level, the distance is measured using the dissimilarity between the OWL entity URIs ($dist_{URI}$):

1. URI dissimilarity for source and target properties
   $f_1(S_P, T_P) = dist_{URI}(S_P.uri, T_P.uri)$

2. URI dissimilarity for source and target property domains
   $f_2(S_P, T_P) = dist_{URI}(class_i, class_j)$,
   $\forall class_i \in domain(S_P), \quad \forall class_j \in domain(T_P)$
3. URI dissimilarity for source and target property ranges
   $f_3(S_P, T_P) = dist_{URI}(class_i, class_j)$,
   $\forall class_i \in range(S_P), \quad \forall class_j \in range(T_P)$

Level 1: Here we calculate the URI dissimilarity for domain and ranges of both the source and target properties, in the same manner as in Level 0. Additionally, we calculate class dissimilarity [Eq. (5)] for both domain and range. The two additional steps are:

1. Compute property domain dissimilarity for all domains
   $f_4((S_P, T_P) = dist_{class}(class_i, class_j)$,
   $\forall class_i \in domain(S_P), \quad \forall class_j \in domain(T_P)$
2. Compute property range dissimilarity for all ranges
   $f_5(S_P, T_P) = dist_{class}(class_i, class_j)$,
   $\forall class_i \in range(S_P), \quad \forall class_j \in range(T_P)$

Finally, we combine all these distances into a combinatorial measure for the complete OWL property dissimilarity measure

$$dist_{property}(S_P, T_P) = \sum_{i=1}^{N} \beta_i f_i(S_P, T_P) \tag{8}$$

where β_i is the weight for a feature i and satisfies the conditions

$$\sum_{i=1} \beta_i = 1 \tag{9}$$

and

$$\beta_i \geq 0 \tag{10}$$

Future Work

The QRMS are generalized representations of actual user queries. We are currently building the application engine that executes QRMs which are matched to a user query. Additionaly, we are working on leveraging our matching algorithm to chain multiple QRMs to create new web interactions.

We are building a statisical translation model to take natural language questions and turn them in to SSQs. We have begun a user study to identify how users can naturally translate their queries into our SSQ model. We anticipate that the user will not need to define the SSQ in the future but the translation will be automatic.

The current implementation does not consider *modifiers* in finding the dissimilariy measure between context properties. If the order of the *modifiers* is changed within a context class, there could be an impact on the meaning of a query. We will incorporate this consideration into our matching algorithms.

In addition, another step is to consider *online execution* as a class similarity matching problem. We can then build classifier systems using the machine learning techniques. Similarly, we can extend the QRM merging process during *offline execution* to a clustering problem and implement conventional statistical methods to solve it. Finally, the parameters α, β, and π mentioned in the equations [Eq. (5), (8), and (2)] are defined manually. We are considering estimation methods that will be able to capture the relevance of the contexts, realm, and modifers of the QRM ontologies.

Conclusion

We have developed ontologies to model user web interaction and user web queries. We have also presented a matching algorithm to discover user web interactions based on a query. Our descriptions are in the OWL language. While we develop our own classes, because of the unified language, our class structure works well with existing ontologies such as *dbpedia* and the *New York Times* corpus. This model will allow question answering systems to use the deep web as a corpus.

References

[1] An, Y. J., Geller, J., Wu, Y.-T., and Chun, S. A. Semantic deep web: automatic attribute extraction from the deep web data sources. In *SAC '07: Proceedings of the 2007 ACM symposium on Applied computing* (New York, NY, USA, 2007), ACM, pp. 1667–1672.

[2] Caverlee, J., and Liu, L. Qa-pagelet: Data preparation techniques for large-scale data analysis of the deep web. *IEEE Trans. on Knowl. and Data Eng. 17*, 9 (2005), 1247–1262. Student Member-Caverlee, James and Member-Liu, Ling.

[3] Ciocoiu, M., and Nau, D. S. Ontology-based semantics. In *KR* (2000), pp. 539–546.

[4] Kim Barrett, B. C. A monotonic superclass linearization for dylan.

[5] Lu, J., Wang, Y., Liang, J., Chen, J., and Liu, J. An approach to deep web crawling by sampling. *Web Intelligence and Intelligent Agent Technology, IEEE/WIC/ACM International Conference on 1* (2008), 718–724.

[6] Madhavan, J., Ko, D., Kot, L., Ganapathy, V., Rasmussen, A., and Halevy, A. Google's deep web crawl. *Proc. VLDB Endow. 1*, 2 (2008), 1241–1252.

[7] Ntoulas, A., Zerfos, P., and Cho, J. Downloading textual hidden web content through keyword queries. In *JCDL '05: Proceedings of the 5th ACM/IEEE-CS joint conference on Digital libraries* (New York, NY, USA, 2005), ACM, pp. 100–109.

[8] Pamuk, M., and Stonebraker, M. Transformscout : finding compositions of transformations for software re-use. Master's thesis, Massachusetts Institute of Technology, 2007.

[9] Steele, Jr., G. L. *Common LISP: the language (2nd ed.).* Digital Press, Newton, MA, USA, 1990.

[10] Straccia, U., and Troncy, R. omap: An implemented framework for automatically aligning owl ontologies. In *Proceedings of SWAP 2005, the 2nd Italian Semantic Web Workshop, Trento, Italy, December 14-16, 2005, CEUR Workshop Proceedings* (2005).

[11] Zhao, H., Meng, W., Wu, Z., Raghavan, V., and Yu, C. Fully automatic wrapper generation for search engines. In *WWW '05: Proceedings of the 14th international conference on World Wide Web* (New York, NY, USA, 2005), ACM, pp. 66–75.