EasyQuerier: A Keyword Based Interface for Web Database Integration System

Xian Li[1], Weiyi Meng[2], Xiaofeng Meng[1]

[1] School of Information, Renmin University of China, {xianli,xfmeng}@ruc.edu.cn
[2] Computer Science Dept., SUNY at Binghamtom, meng@cs.binghamton.edu

Abstract. Recently a lot of work on integrating the search interfaces of multiple Web databases of the same domain into an integrated interface has been reported. Such integrated interfaces enable users to search multiple Web databases using one query. However, there are two potential problems when using these integrated interfaces in practice. First, if the number of domains is large, it may be difficult for users to find the correct domain. Second, the integrated interfaces can become too complicated for ordinary users to use. In this paper, we propose a system called EasyQuerier to tackle these problems. EasyQuerier allows the users to submit keyword-based queries to access the Web databases by first mapping a keyword-based user query to a suitable domain and then translating the user query to a well-formatted query on the integrated interface of the found domain. Our experiments show that both our domain mapping and query translation techniques work very well.

1 Introduction

A large proportion of the information on the Web is stored in the Web accessible databases [1] which are often called *Web Databases* (WDBs). WDB integration is an emerging technique for providing users an unified way to access multiple WDBs. One key research issue here is to automatically integrate the local query interfaces of the WDBs in the same domain into an integrated query interface [2] [3] [4]. Although this issue has received a lot of attention in recent years, using such integrated interfaces in practice has several problems:

1. One integrated interface is able to access only one specific domain. The users need to first determine the desired domain and then find the corresponding integrated interface to submit queries. As the number of domains grows, domain searching becomes an obstacle for the wide use of the integrated interfaces.
2. The integrated query interfaces can be too complex to use for ordinary users because they typically contain a large number of attributes and many of them have lots of pre-defined values.
3. Each attribute in the integrated interface can accept only one value at a time. So a user has to submit multiple queries when he/she wants to set optional search conditions. For example, if a user wants to search a job with job title “DBA” or “Software engineer”, the user has to submit two queries to the integrated interface.

In this paper we propose a novel solution to overcome the above problems while still supporting unified access to multiple WDBs. Our solution provides a simple keyword-based interface "EasyQuerier" plus two mappings, one maps a user query to the correct domain and the other maps the query to one or more queries on the integrated query interface of the domain. EasyQuerier allows a user to submit queries against any domain. Besides, multiple values corresponding to the same attribute on an integrated interface can be entered in the same query. For the job-hunting example given previously, the user can simply enter "DBA or Software engineer".

The rest of this paper is organized as follows. Section 2 provides an overview of EasyQuerier. Section 3 describes our domain mapping solution. Section 4 proposes the query translation algorithm from the keyword-based interface to integrated interfaces. Section 5 reports the experimental results and the analysis. Section 6 reviews related work followed with the conclusion in Section 7.

2 Overview of EasyQuerier

With EasyQuerier, users only need to provide keyword-like queries. Based on the submitted query, the related domain is determined first; then the query is translated into one or more queries that fit the integrated interface of the selected domain; finally each translated query is mapped to the query interfaces of the local Web databases of the domain. In this paper, we focus only on the first two steps of the above process.

In this paper, we assume that an integrated query interface for each domain has already been constructed using some existing techniques (e.g., the WISE-Integrator [3] [5]). EasyQuerier is built on top of these integrated query interfaces. Users can generally submit keyword queries as what they usually do when querying search engines.

Example 1. For the following user query:

Q1: New York or Washington, education, \$2000-\$3000

three keyword units, {New York, Washington}, {education}, and {\$2000-\$3000} (a range) are obtained and their data types are text, text, and money, respectively.

3 Domain mapping

We aim to map a user query to the correct domain automatically without domain information to be separately entered. We first present a model to represent each domain.

3.1 Domain representation model

Our survey covering nine different domains shows that near 90% of the attributes have converging value sets We use the converged value sets to represent each domain. We propose a domain representation model as follows. Specifically, each domain D is modelled by a quadruplet: $D =< d\_ID, CT, AT, VT >$, where

1. *d_ID* is the unique domain identifier.
2. $CT = \{ct_i | i = 1, 2, \cdots\}$ is a set of **C**onceptual **T**erms, which describe the whole domain concept, such as "car", "vehicles", "book", "music CD".
3. $AT = \bigcup_{A \in D} DAL(d\_ID, A_i)$ is a set of **A**ttribute Label **T**erms consisting of attribute labels of the products in this domain. $DAL(d\_ID, A_i)$, **D**omain **A**ttribute **L**abel set, is a set of all the terms related to the attribute label of A_i in domain d_ID. $DAL(d\_ID, A_i)$ consists of terms from three classes: (1)InteLabel: The global label for A_i in the integrated query interface. (2)LocalLabel: All the labels representing A_i in the local query interfaces. (3)OtherLabel: It contains some synonyms

and immediate hypernyms/hyponyms of those terms in InteLabel and LocalLabel obtained using WordNet.

4. $VT = \bigcup_{A \in D} DAV(d\_ID, A_i)$, is a set of the **V**alue **T**erms associated with the products' attributes in the domain d_ID. $DAV(d\_ID, A_i)$, **D**omain **A**ttribute **V**alue set, is a set of all the pre-defined values associated with A_i in domain d_ID.
   For **Character Attribute**, values are classified just like for DAL, i.e., we have InteValue, LocalValue, OtherValue. For **Non-text Attribute**, DAV can be characterized by the pre-defined ranges available on the integrated interfaces.

3.2 Term weight assignment

Often different terms have different ability to differentiate the domains. For example, intuitively attribute label "price" is less powerful than "title" in differentiating the book domain from other domains because the former appears in more domains than the latter. Therefore, we should assign a weight to each term in each domain representation to reflect its ability in differentiating the domain from other domains.

There are different ways to assign weights to a term. In this paper, we adopt a method from [6] that was used in the context of differentiating different component search engines (document databases) in a metasearch. In [6], a statistic called CVV (cue validity variance) is used to measure the skew of the distribution of terms across all document databases, each of which contains a number of documents. For our problem, each domain can be considered as a document database and each local query interface in the domain as a document. Then the CVV of a term can be used as its weight in its ability to differentiate different domains. Denote if_{ij} as the *interface frequency* of term t_j in the i-th domain D_i, i.e., it is the number of times t_j appears in either AT or VT in D_i. Denote CVV_j as the CVV for t_j. Then the weight of t_j in D_i can be computed by: $Weight(D_i, t_j) = CVV_j * if_{ij}$.

3.3 Domain Mapping

After the representation of each domain is generated, we can map each query to a certain domain by computing the similarity between the query and each domain.

We now discuss how to compute the similarity between Q and each domain D. As mentioned in Section 2, we parse a query Q into a set of keyword units $Q = \{u_1, u_2, \cdots, u_n\}$. Therefore, we first compute the similarity between each u_i and the domain D. Each u_i may contain one or more query terms denoted as $\{v_i^1, v_i^2, \cdots\}$. For each v_i^x, we first calculate its similarity with the best matching term in the representation of domain D. Only terms of the attributes that have compatible data types with the data type of u_i are considered. Let T_i^x denote this term set. First, consider the case when v_i^x is a text type query term. The similarity between v_i^x and a term t_j in T_i^x is computed by $Sim(v_i^x, t_j) = \frac{cw}{\max(|v_i^x|, |t_j|)}$, where cw is the number of common words between v_i^x and t_j. Now we consider the case when v_i^x is of a non-text type. In this case, $Sim(v_i^x, t_j)$ is computed based on the percentage of v_i^x that is covered by t_j, i.e., $Sim(v_i^x, t_j) = \frac{|cr|}{|v_i^x|}$, where cr is the shared range between v_i^x and t_j. For both cases, we call the term most similar to v_i^x as v_i^x's *matching term* and denote it as t_i^x.

We now define the similarity between u_i and D, denoted $Sim(u_i, D)$, to be $\max_x\{Sim(v_i^x, t_i^x)\}$. Let t_i^y be the term such that $\max_x\{Sim(v_i^x, t_i^x)\} = Sim(v_i^y, t_i^y)$.

If more than one such t_i^y exist, take the one with the largest $Weight(D, t_i^y)$. Finally, the similarity between Q and D (called the *mapping degree*) is defined as a weighted sum of all the similarities between all the keyword units in Q and D, i.e.,

$$Sim(Q, D) = \sum_{i=1}^{n} Sim(u_i, D) * Weight(D, t_i^y)$$

4 Query Translation

Each query has been parsed into several keyword units before domain mapping. The main challenge in query translation is to map each keyword unit to its most appropriate attribute on the integrated interface of the selected domain. In this section, we first introduce a computation model for query translation, later we discuss how to generate query translation solution based on this model.

4.1 Computation model of the query translation

Definition 4.1 (**K**eyword-**A**ttribute **M**atching (KAM)). Given a keyword unit u and an attribute A from the integrated interface, their mapping is denoted as $KAM(u, A)$.

Definition 4.2 (**D**egree of **M**atching (DM)). DM is the degree of matching for a KAM, with value range [0, 1]. Given k keyword units and m attributes, $k * m$ KAMs can be generated and their DM values form a $k*m$ matrix, which will be called the DM matrix.

Definition 4.3 (**Q**uery **T**ranslation **S**olution (QTS)). A QTS represents a strategy of filling in the query interface. A QTS is comprised of k KAMs, where k is the number of keyword units.

Definition 4.4 (Conviction). This measurement determines whether a QTS is reasonable. The larger the DM of a KAM, the more reasonable the KAM is. Thus, the QTS containing such a KAM will more likely yield sounder query translation. Thus the value of Conviction is computed as a weighted sum of all the related DMs.

4.2 Computation of DM

In our system, $DM(u_i, A)$ is determined by the similarity between the keyword unit u_i and the value set of attribute A. The value set of A on the integrated interface of domain d_ID is DAV(d_ID, A) (see Section 3.1).

A keyword unit in EasyQuerier may contain more than one keyword related to the same attribute. Let u_i = $\{v_i^1$ or v_i^2 or $\cdots$ or $v_i^p\}$ be such a keyword unit. When computing the DM of a $KAM(u_i, A_j)$, we first calculate $Sim(v_i^x, A_j)$ which represents the similarity between a value v_i^x and an attribute A_j, then the maximum of all the similarities is the value of $DM(u_i, A_j)$. For each t_j in the DAV of A_j $Sim(v_i^x, t_j)$ is computed as what mentioned in section 3.3. $Sim(v_i^x, A_j)$ is the maximum value of all the $Sim(v_i^x, t_j)$.

Finally, the $DM(u_i, A_j)$ is aggregated from all the Sim values related to the keywords in u_i using $DM(u_i, A_j) = \max_{x=1}^{p}\{Sim(v_i^x, A_j)\}$.

4.3 Computation of Conviction and QTS Generation

In Definition 4.4, the Conviction value of a QTS is a weighted sum of the DMs of the related KAMs. We compute a weight $w(A_j)$ for each attribute A_j based on its *interface frequency*. Let if_i be the number of local query interfaces that contain attribute

A_i. Intuitively, if an attribute appears in more local interfaces of a domain, it is more important in the domain. Based on this, we compute $w(A_j) = if_j/(\sum_i if_i)$. Finally, for QTS = $KAM(u_1, A_1^{'}) \wedge KAM(u_2, A_2^{'}) \wedge \cdots \wedge KAM(u_k, A_k^{'})$, we use the following formula to compute its conviction:

$$Conviction(QTS) = \sum_{i=1}^{k} w(A_i^{'}) * DM(u_i, A_i^{'})$$

5 Experiments

A prototype of EasyQuerier has been implemented. The data collection for the experiment includes: web databases and user queries. (1) Web databases: WDBs covering 9 different domains are collected with 50 databases for each domain. (2) User query collection: 10 students across five different majors are invited as the evaluators of our demo system. For each domain, every student provides two different keyword queries.

The evaluation for both domain mapping and query translation is similar: we identify a *correct mapping/translation* by checking whether the selected domain/translated query with the largest similarity matches the user's intention. If the user is not satisfied with the top result, we let them click the button "more" for more choices In general, the top 3 choices are provided. If the correct result appears in these choices, we consider the result an *acceptable mapping/translation*; otherwise the mapping/translation is considered to be *wrong*.

Results on domain mapping. The experiment on domain mapping is conducted on the 9 domains. For each query, the produced domains are ranked in descending order of their similarities with the query.

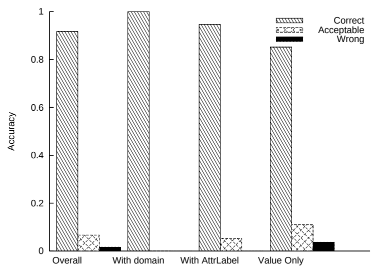

Fig.1. Domain mapping accuracy

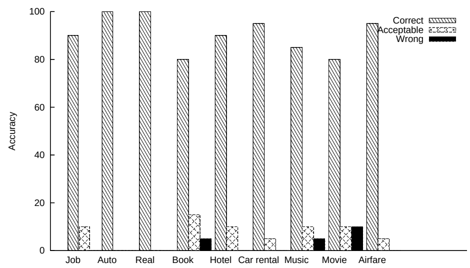

Fig.2. Query translation accuracy

Figure 1 shows the overall percentages of the mapping results that are correct, acceptable and wrong, respectively, for all queries as well as for each group of queries. As it can be seen, the overall accuracy is very good. Failurs are mostly caused by inadequate information in user queries.

Results on query translation. After translating the source query, one or more translated queries are generated. Figure 2 shows the percentages of the translations that are correct, acceptable and wrong for each domain. We find that for the nine domains considered, most queries can be translated correctly. However, for the book, music and movie domains, the average accuracy is lower at about 82.5%. The main cause of failures for these domains is that many important attributes such as "title", "author", "singer", and "director" are textboxes for which building a value set is difficult.

6 Related Work

Automatic interface integration has been a hot issue in recent years. WISE-integrator [3] and Meta-Querier [2] aim at integrating the complex query interfaces provided by WDBs. As discussed in Section 1 these integrated query interfaces are likely to be too complex for ordinary users and our work aims to provide an easy-to-use interface.

Our work is related to researches that translate natural language queries to structured queries (such as SQL) to support natural language access to structured data (e.g., [7][8]). The main differences between these works and our work reported here are as follows. First, they do not deal with the domain mapping problem while we do. Second, they deal with mostly relational databases while we deal with Web query interfaces. Third, they have access to both the schema information and the actual data but we only have access to the schema and very limited pre-defined values available on the query interface but do not have access to the full data. Finally, we deal with keyword queries rather than real natural language queries.

7 Conclusion

In this paper, we proposed a novel keyword based interface system EasyQuerier for ordinary users to query structured data in various Web databases. We developed solutions to two technical challenges, one is how to map keyword query to appropriate domains and the other is how to translate the keyword query to a query for the integrated search interface of the domain. Our experimental study involving real users showed that our solutions can produce very promising results.

Acknowledgment. This work is supported in part by the NSF of China under grant #s 60573091, 60273018; NSF of Beijing under grant #4073035; Program for New Century Excellent Talents in University (NCET); US NSF grants IIS-0414981 and CNS-0454298.

References

1. BrightPlanet: The deep web: Surfacing hidden value. (http://brightplanet.com)
2. Chang, K.C.C., He, B., Zhang, Z.: Toward large scale integration: Building a metaquerier over databases on the web. In: CIDR. (2005) 44–55
3. He, H., Meng, W., Yu, C.T., Wu, Z.: Wise-integrator: An automatic integrator of web search interfaces for e-commerce. In: VLDB. (2003) 357–368
4. Dragut, E.C., Wu, W., Sistla, A.P., Yu, C.T., Meng, W.: Merging source query interfaces on web databases. In: ICDE. (2006) 46
5. He, H., Meng, W., Yu, C.T., Wu, Z.: Wise-integrator: A system for extracting and integrating complex web search interfaces of the deep web. In: VLDB. (2005) 1314–1317
6. Yuwono, B., Lee, D.L.: Search and ranking algorithms for locating resources on the world wide web. In: ICDE. (1996) 164–171
7. Androutsopoulos, I., Ritchie, G.D., Thanisch, P.: Natural language interfaces to databases - an introduction. CoRR **cmp-lg/9503016** (1995)
8. A. Popescu, O.E., Kautz, H.: Towords a theory of natural language interfaces to databases. International Conference on Intelligent User Interfaces. (2003)