

MetaQuerier: Querying Structured Web Sources On-the-fly*

Bin He, Zhen Zhang, Kevin Chen-Chuan Chang
Computer Science Department
University of Illinois at Urbana-Champaign
{binhe, zhang2}@uiuc.edu, kcchang@cs.uiuc.edu

1. INTRODUCTION

Recently, we witness the rapid growth and thus the prevalence of databases on the Web. Our recent survey [2] in April 2004 estimated 450,000 online databases. On this deep Web, myriad online databases provide dynamic query-based data access through their *query interfaces*, instead of static URL links. As the door to the deep Web, it is essential to integrate these query interfaces for integrating the deep Web.

However, while there are myriad useful databases online, users often have difficulties in first *finding* the right sources and then *querying* over them. Consider user Amy, who is moving to a new town. To start with, different queries need different sources to answer: Where can she look for real estate listings? (*e.g.*, *realtor.com*.) Studying for a new car? (*cars.com*.) Looking for a job? (*monster.com*.) Further, different sources support different query capabilities: After source hunting, Amy must then learn the grueling details of querying each source.

To enable effective access to databases on the Web, since April 2002, we have been building a “metaquerying” system, the *MetaQuerier* (*metaquerier.cs.uiuc.edu*). Our goal is two fold— First, to make the deep Web systematically *accessible*, it will help users *find* online databases useful for their queries. Second, to make the deep Web uniformly *usable*, it will help users *query* online databases.

Such deep-Web integration faces new challenges— for coping with the *large scale*: The deep Web is a large collection of queryable databases (well on the order 10^5 , as mentioned earlier). As the large scale mandates, first, such integration is *dynamic*: Since sources are proliferating and evolving on the Web, they cannot be statically configured for integration. Second, it is *ad-hoc*: Since queries are submitted by users for different needs, they will each interact with different sources— *e.g.*, in Amy’s case: those of real estates, automobiles, and jobs. Thus, toward large-scale integration, the MetaQuerier must achieve the requirement of *on-the-fly integration*— That is, the MetaQuerier must mediate user’s queries on-the-fly for relevant sources, with no pre-configured *per-source* (*i.e.*, source-specific) knowledge.

*This material is based upon work partially supported by NSF Grants IIS-0133199 and IIS-0313260. Any opinions, findings, and conclusions or recommendations expressed in this publication are those of the author(s) and do not necessarily reflect the views of the funding agencies.

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage, and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.

SIGMOD 2005 June 14-16, 2005, Baltimore, Maryland, USA.
Copyright 2005 ACM 1-59593-060-4/05/06 \$5.00.

Toward such on-the-fly integration, it is essential to develop techniques to enable *on-the-fly query translation* across different deep Web sources, which means we can translate queries to a source (*e.g.*, *amazon.com*) without having specifically prepared translation knowledge for that source. In this demo, we will illustrate our new techniques to realize such on-the-fly query translation. In particular, we will show how to translate queries between two “unseen” sources— That is, given two new query interfaces we have never seen before, although we do not have any translation knowledge between them, we are able to translate queries from one interface to another with our on-the-fly translation techniques.

Such on-the-fly query translation poses a significant new challenge on the generality and extensibility of the translation framework. Existing work [1] pursues a *source-based rule-driven* framework and thus cannot satisfy such requirements. In contrast, we propose a generic *type-based search-driven* translation framework by leveraging the “regularities” across the implicit data types of query constraints [8]. We observe that query constraints of different concepts (*e.g.*, the concept about title and the one about author) often share similar patterns. This observation indicates an implicit notion of *data type* that a locality of translatable patterns are often used by concepts with the same data type (*e.g.*, text or numeric). Therefore, instead of source-based knowledge, we can “encode” more generic translation knowledge for each data type. To realize this type-based translation, we abstract query translation as a search problem and thus develop an extensible *search-driven* mechanism.

This demo proposal is built upon our earlier works on interface extraction [9] and schema matching [4, 5], which have been demonstrated in SIGMOD 2004 [6]. As a research project, we take a “divide and conquer” approach in developing the MetaQuerier by identifying and studying key tasks step by step. Our past efforts mainly focus on discovering matchings among attributes from a set of Web query interfaces for each domain, which consists of interface extraction (*i.e.*, extracting the attribute information, given a query interface in HTML format) and schema matching (*i.e.*, discovering the semantic correspondences among attributes in interfaces). Built upon the discovered matchings, this demo thus enables the on-the-fly translation of a user’s query between interfaces.

2. ON-THE-FLY QUERY TRANSLATION

We position our on-the-fly query translation problem as *translating a query in a source interface to a target interface without source-specific translation knowledge*. In particular, as Figure 1 illustrates, given a specific *source query* Q_s instantiated from a source interface QI_1 , and a *target interface* QI_2 , query translation maps among semantically related predicates (as given by a schema matcher), and outputs a target query Q_t which is “close” (*w.r.t.* a given semantic closeness metric \mathcal{C}) to Q_s and “expressible” against

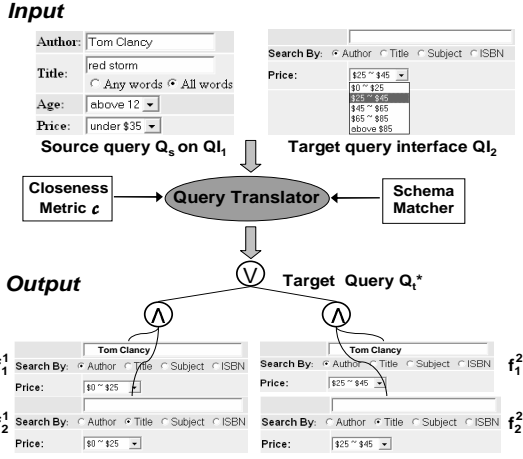


Figure 1: Query translation: An overview.

the target interface. As Figure 1 shows, such a translation forms a Boolean network, with Boolean operators (e.g., \wedge , \vee) connects “leaf queries” (e.g., f_1^1 , f_1^2 , f_2^1 and f_2^2 as different fillings of forms), which are supported by the target interface. After generating such a translation, the translator can thus execute the query by fulfilling those leaf queries through the interface, and further combine individual query results using Boolean operators to complete the translation. As discussed in Section 1, such translation must happen on-the-fly, i.e., without pre-configured translation knowledge for sources.

While critical for integrating Web databases, such on-the-fly translation imposes great challenges. To start with, at the “macro” level: As the translation needs to deal with both *semantics* - to generate a semantically close translation, and *query capability* - to generate an expressible one, what is the right framework for interleaving these two goals? Further, at the “micro” level: At the core of query translation, semantic mapping boils down to mapping between matching predicates. How to realize such mapping without pre-configured knowledge?

To begin with, let us discuss the challenge in “macro” part— That is, *how to realize the two goals of semantic closeness and expressibility at the same time?* A baseline framework is thus to search among all expressible translations for the semantically closest one. However, such a framework is too prohibitive to be realistic because the scope of search is too huge. We thus need to develop a realistic approach, which has a more focused search scope, while still achieves the two goals.

Intuitively, we want the search to happen within a minimal scope, e.g., for individual predicates. In particular, instead of searching the translation space of the entire query Q_s , we hope to focus on each queried predicate individually. For instance, we can search for the closest translation for only the predicate on *price* in Figure 1. For the complete query Q_s , we expect to get its correct translation by combining the closest translations for all the predicates. This intuition has great advantages: 1) It significantly reduces the scope of search by removing the combinatorial effects of predicate combinations. 2) It reduces the scopes of semantic knowledge as needed for handling semantic mapping. We can formally prove that combining predicate-level semantic searches is equivalent to the baseline framework in terms of satisfying the two goals. (For space limitation, we omit the proof in this demo proposal.)

To concretely realize such an idea, we build a *semantic-then-syntax* framework, which consists of two steps, *semantic mapping* and *syntax construction*, as Figure 2 illustrates. In particular, semantic mapping searches for the semantically closest translation

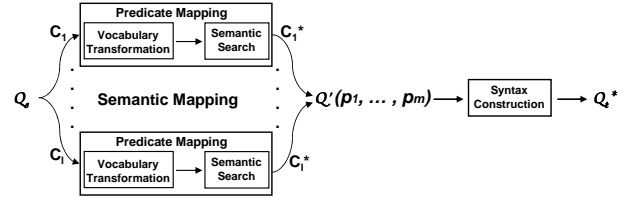


Figure 2: Semantic-then-syntax translation framework.

for each predicate in the source query Q_s individually. Syntax construction then combine the translated predicates to form queries on the target interface, which satisfy the expressibility. Next, we discuss these two steps in more details respectively.

Semantic Mapping: More formally, given a source query $Q_s = c_1 \wedge \dots \wedge c_l$, where c_1, \dots, c_l are predicates, semantic mapping generates a closest translation $Q' = c_1^* \wedge \dots \wedge c_l^*$ by mapping each predicates c_i to c_i^* separately. For instance, consider the translation scenario in Figure 1. As input, suppose we are given a source query as $Q_s = [\text{author}; \text{contain}; \text{Tom Clancy}] \wedge [\text{title}; \text{contain}; \text{red storm}] \wedge [\text{price}; \leq; 35]$. We are also given a closeness metric \mathcal{C} as minimal subsuming translation. The semantic mapping will map each of the three predicates into corresponding predicates in the target interface, which satisfy \mathcal{C} . In particular, $[\text{price}; \leq; 35]$ in Q_s is mapped to $[\text{price}; \text{between}; 0,25] \vee [\text{price}; \text{between}; 25,45]$ in the target predicates. The closest translation Q' is thus $[\text{author}; \text{contain}; \text{Tom Clancy}] \wedge [\text{title}; \text{contain}; \text{red storm}] \wedge ([\text{price}; \text{between}; 0,25] \vee [\text{price}; \text{between}; 25,45])$.

The challenge of such semantic mapping is that because we do not have translation knowledge customized to specific sources, how can we realize an on-the-fly mapping machinery? In particular, our machinery must define: First, a *general scope* to encode the translation knowledge, which is not specific to any source; Second, a *general mechanism* to apply the knowledge, which is suitable for the scope. To establish the scope of translation, we conducted a survey, which shows that there exists *localities* of translation across different predicate templates. Not surprisingly, we find that such translation localities are consistent with the notion of *data types*. This observation on data types motivates a type-based translation framework, which encodes the translation knowledge in the scope of types (i.e., localities) instead of per-source as opposed to the traditional framework.

Further, the type-based knowledge encoding gives a platform to compare the semantic closeness of different translations. For instance, a numeric data type allows us to compare the closeness of two numeric ranges (e.g., $[\text{price}; \leq; 0,35]$ vs. $[\text{price}; \text{between}; 25-45]$) by mapping them against a numeric line and compare their coverage. Therefore, types-based translation enables a *search-driven* approach to explore and compare different semantic mappings and looks for a closest one.

In our current development, we assume only simple 1:1 mappings, i.e., a source predicate will only be matched and thus translated to one target predicate template. Although mapping in general can be complex (e.g., $\text{lastname+firstname:author}$), simple 1:1 mappings are the most common cases. Our experiment on real query interfaces also justifies this observation— It shows that considering complex mapping only marginally improve the translation accuracy.

Syntax Construction: After semantic mapping generates a closest translation Q' , the syntax construction step takes care of the expressibility of querying Q' on the target interface. For instance, the Q' generated above cannot be directly issued on the target interface QI_2 , since the expressibility of QI_2 can only afford to query either *author* or *title*, but not both. Therefore, we need to transform Q' into a Boolean expression of subqueries, which satisfy the

Author:	<input type="text"/>	Title:	<input type="text"/>
Price:	<input type="text" value="\$25 ~ \$45"/>	Price:	<input type="text" value="\$25 ~ \$45"/>

(1). Conjunctive Form F_1 (2). Conjunctive Form F_2

Subject:	<input type="text"/>	ISBN:	<input type="text"/>
Price:	<input type="text" value="\$25 ~ \$45"/>	Price:	<input type="text" value="\$25 ~ \$45"/>

(3). Conjunctive Form F_3 (4). Conjunctive Form F_4

Figure 3: The virtual forms of QI_2 .

expressibility of QI_2 .

To begin with, we transform Q' into a Boolean expression $C_1 \vee C_2 \vee \dots \vee C_m$, where all the C_i s consist a conjunction of predicates over the same set of attributes. For instance, the Q' in our running example is transformed into $C_1 \vee C_2$ where $C_1 = [\text{author}; \text{contain}; \text{Tom Clancy}] \wedge [\text{title}; \text{contain}; \text{red storm}] \wedge [\text{price}; \text{between}; 0,25]$, and $C_2 = [\text{author}; \text{contain}; \text{Tom Clancy}] \wedge [\text{title}; \text{contain}; \text{red storm}] \wedge [\text{price}; \text{between}; 25,45]$. C_1 and C_2 have the same sets of attributes **author**, **title**, and **price**.

Now the problem becomes how to express each C_i according to the target interface. In particular, as we can see from Figure 1, the interface QI_2 can only query one of the attributes **author**, **title**, **subject** and **ISBN** at one time. QI_2 is thus equivalent to giving the user four separate *virtual forms* F_1, F_2, F_3 and F_4 , where each form is a set of conjunctive conditions, as Figure 3 shows. Therefore, our problem can be formally phrased as: Given a query C_i as a conjunctive query, how to rewrite it as a boolean expression $D_1 \wedge D_2 \wedge \dots \wedge D_r$, where each D_j can be expressed using one of the virtual forms of the target interface.

The number of answers to this problem is in fact infinite. For instance, C_1 can be transformed as $D_1 \wedge D_2$, where $D_1 = [\text{author}; \text{contain}; \text{Tom Clancy}] \wedge [\text{price}; \text{between}; 0,25]$, and $D_2 = [\text{title}; \text{contain}; \text{red storm}] \wedge [\text{price}; \text{between}; 0,25]$. As we can see D_1 and D_2 can be expressed using F_1 and F_2 respectively. However, C_1 can also be transformed as $D_1 \wedge D_3$, where D_1 is the same as above and $D_3 = [\text{title}; \text{contain}; \text{red storm}]$, which can also expressed using F_2 . Similarly, we can construct many other answers such as $D_1 \wedge D_2 \wedge D_3$. Therefore, we need to choose one among all the possible ones as our best answer.

In particular, we take a greedy approach, which iteratively chooses a virtual form that maximally covers the predicates until all predicates are covered. Further, to maximize the usage of a form, if multiple forms covers the same set of uncovered predicates, we choose the one with maximal number of predicates. For instance, for transforming C_1 , we first choose F_1 , which maximally covers **author** and **price**, and leaves **title** uncovered. In the next step, we choose F_2 , which maximally covers **title** and **price**. Now all the predicates are covered, and the generated translation is thus $D_1 \wedge D_2$. Similarly, we transform C_2 into $D_4 \wedge D_5$, where $D_4 = [\text{author}; \text{contain}; \text{Tom Clancy}] \wedge [\text{price}; \text{between}; 25,45]$, and $D_5 = [\text{title}; \text{contain}; \text{red storm}] \wedge [\text{price}; \text{between}; 25,45]$. Therefore, the final translation of the source query Q_s is $(D_1 \wedge D_2) \vee (D_4 \wedge D_5)$.

The reason we apply the strategy of maximal form usage among multiple virtual forms is two-fold: First, by choosing the form with the most predicates, the generated query (e.g., D_2) is the most restrictive (compared with D_3), and thus the cost for retrieving the query results is minimal. Second, a more important reason for choosing a maximal form is due to the “binding” requirement of query interfaces [7]– That is, some attributes in query interfaces cannot be queried alone and must be bounded with other attributes. For instance, the **price** in QI_2 may have to be bounded with either **author** or **title**. By choosing the maximal forms in syntax construction, we can prove a good property: If our translation fails in querying the target interface due to the binding requirement, no

other translations can succeed. (Due to the space limitation, we omit the proof in this demo proposal.)

3. DEMONSTRATION

The implementation of the system adopts Python 2.3, wxPython 2.4 and Javascript on Windows XP. Javascript and Python are used to implement the model construction function, which builds source query models of interfaces. The main algorithms of query translation is implemented with Python and the graph user interface is implemented in wxPython 2.4.

This demo illustrates an automatic process to translate queries between two interfaces on the fly. It starts with a source and a target query interface specified by the user, and asks user to fill in a query in the source interface. The system then automatically translates the source query and fills in the target interface. The query translation happens on-the-fly because there is no source specific knowledge prepared for the two sources chosen by the user. We demonstrate such on-the-fly translation on the TEL-8 dataset of the UIUC Web Integration Repository [3]. TEL-8 dataset contains the original query interfaces of 447 deep Web sources from 8 representative domains.

The demonstration proceeds in three steps: First, *query specification* asks the user to choose a source and a target interface for the system to work on. The user then fills in the source interface to formulate a source query. Second, *model construction* extracts the query models of two interfaces, presents them using internal representation and then matches attributes between these two models. Our past efforts on interface extraction [9] and schema matching [4, 5] are used to extract query models and match attributes respectively. Finally, *query translation*, the core of this demo, applies the semantic-then-syntax translation framework discussed in Section 2 to on-the-fly translate the source query on the target model and further fills in values in the target interface.

4. REFERENCES

- [1] K. C.-C. Chang and H. Garcia-Molina. Mind your vocabulary: Query mapping across heterogeneous information sources. In *SIGMOD Conference*, 1999.
- [2] K. C.-C. Chang, B. He, C. Li, M. Patel, and Z. Zhang. Structured databases on the web: Observations and implications. *SIGMOD Record*, 33(3):61–70, September 2004.
- [3] K. C.-C. Chang, B. He, C. Li, and Z. Zhang. The UIUC web integration repository. Computer Science Department, University of Illinois at Urbana-Champaign. <http://metaquerier.cs.uiuc.edu/repository>, 2003.
- [4] B. He and K. C.-C. Chang. Statistical schema matching across web query interfaces. In *SIGMOD Conference*, 2003.
- [5] B. He, K. C.-C. Chang, and J. Han. Discovering complex matchings across web query interfaces: A correlation mining approach. In *SIGKDD Conference*, 2004.
- [6] B. He, Z. Zhang, and K. C.-C. Chang. Knocking the door to the deep web: Integrating web query interfaces. In *SIGMOD Conference, System Demonstration*, 2004.
- [7] R. Yerneni, C. Li, H. Garcia-Molina, and J. D. Ullman. Computing capabilities of mediators. In *SIGMOD Conference*, 1999.
- [8] Z. Zhang, B. He, and K. C.-C. Chang. On-the-fly constraint mapping across web query interfaces. In *Proceedings of the VLDB Workshop on Information Integration on the Web*, 2004.
- [9] Z. Zheng, B. He, and K. C.-C. Chang. Understanding web query interfaces: Best-effort parsing with hidden syntax. In *SIGMOD Conference*, 2004.