Translating Naive User Queries on the Semantic Web

Baoshi Yan, Robert MacGregor
Distributed Scalable Systems Division
Information Sciences Institute
University of Southern California
{baoshi,macgregor}@isi.edu

ABSTRACT

Query is an important way of information retrieval. One type of queries is those to search engines, which are lists of keywords without structures. Another type of queries is those to databases or knowledge bases, which must conform to the structure and terminology of the data source (e.g., SQL query to a database). In this paper, we deal with another type of queries: *naive user queries*–queries in users' own terms and structures. We envision that this type of queries would be a common phenomenon on the future Semantic Web. We propose an approach that, given a naive user query, translates it into a list of queries conforming to different data source schemas. The approach is based on partial alignment between some data sources. An early prototype showed that the result is promising.

1. INTRODUCTION

When people want to retrieve information, they usually issue a query. The most used form is queries to search engines, which is a combination of keywords. However, people cannot specify semantic structure between these keywords. This is more due to that search engines mostly deal with natural language texts that are hard to extract semantic structures from. On the other hand, databases and knowledge bases have semantic structures, but they require queries (e.g., a SQL query to a DB, or an ASK function to a KB) to conform to their terminology and structures. On the future Semantic Web[6], neither form of queries is sufficient. Search-engine-style queries don't impose restrictions on the terms used, but don't have semantic structure either, which put them in a disadvantaged situation in a web of semantic structures. Queries in DB or KB style have to conform to the schema used by individual data sources. On the Semantic Web, there'll be numerous data sources with different schemas. Requiring people to write different queries for individual data sources is a daunting task.

Thus we propose that it is necessary to deal with another type of user queries: *naive user queries*–queries in users' own terms and semantic structures. Instead of letting information seekers understand the schema used by an information provider, we could shift the load to the information provider to understand naive user queries.

Without losing generality, we represent a naive user query as a list of triple patterns (s,p,o)[1]. Table 1 lists some examples. We represent data source schemas as RDFS schemas [1](or ontologies). Equivalent queries can be written with different terms, as can be seen from first two queries in the table. Semantic structures between terms are binary relations: p is the kind of relationship between s and o. The triple patterns roughly conform to RDF [3] data model except that the terms here are users' descriptions, not URI's (Universal Resource Identifiers). The triple patterns allow people to specify semantic structures, and we think it keeps simple enough so that ordinary users can write it. It is also not hard to come up with a GUI to automatically add syntax sugar with little user effort.

Such type of user queries might not conform to the schemas of available data sources. In the rest of the paper, we discuss an approach to translate such queries into equivalent ones conforming to actual data source schemas.

2. PROBLEM FORMULATION

Translating naive user queries is difficult. This is because people could use all kinds of terms and structures to represent the same thing. However, we have two observations that would greatly ease the problem.

The first observation is that the variation of terms and structures used for a concept is much less than the variation of data schemas and queries. The huge variations of schemas or queries are mainly due to the combinatorial explosion of variations of terms and structures. For example, "moviename" and "movietitle" are two terms to represent a movie name; "rating" and "movierating" are two terms to represent rating of a movie. Their combination could lead to four data schemas. If we take into account that RDFS schemas qualify each term with URI's (so there might exist "http://com1#moviename", "http://com2#moviename"), then the number of schemas is infinite.

In a class at the University of Southern California, students are required to create a RDFS schema at their choices as a Semantic Web assignment. Seven of them happened to choose movie-related schemas. Table 2 shows parts of these seven schemas, in which many term duplicates can be seen.

The second observation is that among the terms used to represent the same concept, a few number of terms are used more often than others. The 20-80 law seems to apply here, which implies that a small number of terms will account for most usages.

[1]Although syntax doesn't affect our discussion, we use RDQL-alike [2] syntax for convenience.

.

#	Naive User Query	Meanings
1	select ?2 where (?1, moviename, "The Ring")(?1, director, ?2)	Find the director of the movie "The Ring"
2	select ?2 where (?1, title, "The Ring")(?1, directorname, ?2)	Find the director of the movie "The Ring"
3	select ?2 where (?1, type, ThrillerMovie) (?1, name, ?2)	Find all the thriller movies in the data source

Table 1: Some Naive User Query Examples

The above two observations led us to believe that, if we could accumulate a number of variations of terms and structures, we might be able to translate a great deal of naive user queries by matching their combinations against the given query. This belief is further strengthened by the fact that naive user queries tend to be short and simple.

Although it is possible that terms accumulated in one domain might help translating naive queries in other domains, we'll focus on only one domain in this paper. We are trying to solve the following kind of problems.

Problem Formulation:

Data A list of data schemas in the same domain, and some alignments between some properties and classes in some schemas (i.e., partial alignments).

For the examples in this paper, we use the seven movie schemas. Among the properties and classes specified in Table 2, we aligned the properties and classes in Schema 1 with those in Schema 6, and those in Schema 4 with those in Schema 5. This is all the alignment information we use throughout the paper.

Input Any given naive user query.

In the examples described in this paper, the given naive query is (?1, name, "The Ring") (?1, moviedirector, ?3) (?1, type, ThrillerMovie).

Output Translations of the given user query into each individual schema.

Goal The goal is to find as much correct translations as possible. We can use concepts of precision and recall from information retrieval as metrics. Precision measures the percentage of correct translations among all the generated translations. Recall measures how many correct translations are generated by the algorithm out of all possible correct translations.

We'll talk about how we handle this problem in the next section.

3. APPROACH

We start with two not-so-satisfactory approaches we've tried and then introduce our adopted approach. For illustration purpose, let's assume we are given a user query $(?1, name, ''TheRing'')(?1, moviedirector, ?2)$ and try to translate it into the movie schemas we have.

The first approach is to build a global schema. Just think of each column in the Table 2 as defining a concept in the global schema, with all the cells in the column as possible labels of the concept. Translation with this global schema is easy. However, constructing and maintaining such a global schema is difficult. Also, it is hard to represent complex alignments (e.g., one's parent whose gender is female is one's mother) in the global schema.

Another approach we tried without constructing a global schema is to map the problem into a path searching problem. The idea is to construct a graph $G(V, E)$ with V being the node set representing all the classes and properties from all schemas, and $E = E_1 \bigcup E_2$ where $E_1 = \{(v_1, v_2)|v_1 \approx v_2\}$, $E_2 = \{(v_1, v_2)|v_1$ and v_2 belong to the same schema$\}$. $v_1 \approx v_2$ means that there exist some kind of alignment between node v_1 and v_2. Given such a graph, we could look up the node (or a set of nodes) v_1 with label "name" and the node v_2 with label "moviedirector", and then we compute the paths [18] between v_1 and v_2. The intuition here is that if there's no path between v_1 and v_2 then there's no schema that would contain a translation of the query. After some post-processing on the resulted paths we could infer possible translations of the original query into different schemas. The problem with this approach is still the difficulty of representing complex alignments.

The approach we finally used is based on query rewriting. The intuition is to rewrite the original query based on available alignments. Then we check resulted translations to pick out those making sense. The algorithm works as following:

Decomposition Step: Decompose the original query into individual triple patterns t_1, t_2....

Query Rewriting Step: For each triple pattern $t = (s, p, o)$, find all possible rewritings of it according to the following rules. Repeat this step until no more new rewritings are produced.

Exact Name Matching: if p is a local name[2], find all properties whose local names match. If p is already a property name, find all properties with the same local name. For each matched property name p_i, produce a rewriting $t_i = (s, p_i, o)$. If p is a type predicate name, it indicates that o is a class name. Do the same thing to find out all classes with the same local name. For each matched class name C_j, produce a rewriting $t_j = (s, p, C_j)$.

Approximate Name Matching: if p is a local name, we also find all properties whose local name will match p after some manipulations. The manipulations are on local names of a property and its domain classes (i.e., the classes it adheres to). For example, if a property's local name is "movieName" and its domain class' local name is "Movie", then it matches the p of "name". If its local name is "name", then it matches "movieName", "hasName", and "hasMovieName". If p is a property name, we produce several variations of its local name and look for other properties with the same local name as those variations. We are not afraid of producing meaningless local names, because there won't be properties matching the meaningless names anyway. Note that some other approximate name matching techniques could

[2]In RDFS, a property is normally represented as a URI such as http://movie.org#movieName. The part after # is so-called local name.

also be used, such as the one based on edit distance [11]. As previous, we'll produce some new rewritings.

DESCRIPTION MATCHING: if p is a long string, it would be desirable to compare p with property descriptions. We haven't implemented this yet.

ALIGNMENT: We represent alignments as query rewriting rules. For example, the alignment between "movieName" in Schema 1 and "title" in Schema 3 is represented as $(?1, S1 : movieName, ?2) \leftrightarrow (?1, S3 : title, ?2)$. The "ThrillerMovie" class in S1 is aligned to the "Movie" class in Schema 5 with a "movieGenre" property of value "Thriller": $(?1, type, S1 : ThrillerMovie) \leftrightarrow (?1, type, S5 : Movie)(?1, S5 : movieGenre, ''Thriller'')$. Triple patterns matching either side of an alignment rule would result in a rewriting based on the other side of the rule.

INFERENCE: For a query on a class, its subclasses also matchthe query, i.e., for each C_1 that is a subclass of C_2 we have $(?1, type, C_2) \rightarrow (?1, type, C_1)$. Here $\rightarrow$ is not logical inference; it represents translation direction. For example, a "ThrillerMovie" matches a query looking for a "Movie". We expect that other kinds of inference rules could also be used.

As an example, Table 3 shows a sample series of rewritings starting with triple pattern $(?1, name, '' TheRings'')$.

Pruning Step 1: As a result of the last step, we'll have a list of rewritings for each individual triple pattern in the original query. Note that because of the use of alignment and inference rules, a rewriting for an original triple pattern might contain more than one triple pattern. Thus a rewriting might contain properties and classes from different schemas. Such rewriting doesn't make sense in the final answer, thus is removed. We'll illustrate this with examples in Table 4 later.

Pruning Step 2: For each schema used in the rewritings, we check whether all the original triple patterns have rewritings in that schema. If the answer is no and partial translation is not allowed, we could prune all the rewritings in that schema. We'll illustrate this with examples in Table 5 later.

Checking Step: Now for each schema left, for each original triple pattern we pick a rewriting of it in the schema. The picked rewritings form a possible translation of the original user query in the schema. We check the semantic structure of translation against that of the schema; the translation is removed if semantic structures don't match. We'll illustrate this with examples in Table 5 later.

The checking step is crucial. In the query rewriting step we try to come up with as many translations as possible, and the checking step ensures that wrong translations are actually removed and final answers make sense.

Let's illustrate our approach with one experiment. The data we used and the query we faced are those mentioned in the "Problem Formulation" of Section 2.

Table 4 shows that the rewriting $(?1, S4 : movieDirector, ?3)$ $(?3, S5 : personName, ??3)$ for input query pattern $(?1, moviedirector, ?3)$ is pruned in this step, which is because the rewriting contains information from both Schema 4 and Schema 5. Thus the rewriting doesn't make sense as a part of the final translation.

Table 5 shows the result of Pruning Step 2 and Checking Step. The first translation in the table is pruned because it doesn't have a rewriting for the input triple pattern $(?1, type, ThrillerMovie)$. The third translation is pruned because its rewriting $(?1, S6 : Director, ?3)(?3, S6 : Title, ??3)$ doesn't match the semantic structure of Schema 6. In Schema 6 "S6:Title" is a property of the "S6:Movie" class and the value of the "S6:Director" property is a string which cannot have properties. Similarly, the fourth translation is pruned because an "S5:Movie" cannot have an "S5:personName" property.

The experiment turned out to be successful. It didn't contain erroneous translations, and found all correct translations. However, we have yet to prove that our approach will work in real life. This is more due to that we lack real data and real users to experiment with. Nevertheless, the preliminary results with our current (very limited) data are encouraging.

Another desirable feature of our approach is that the whole architecture is extensible. Different kinds of knowledge, from exact name matching to inference rules, can be utilized in the query rewriting step. The ability to incorporate inference rules is especially important for the Semantic Web.

4. EVALUATION

The ideal evaluation of our approach would be to test it with a number of real user queries, and to measure the performance with metrics like precision and recall as described in Section 2. It would also be helpful to adjust the number of alignments in the knowledge base and to see how the precision and recall change accordingly. However, collecting a large number of real user queries is difficult. Instead, for evaluation purpose, we searched for other movie schemas on the web, and use them construct naive user queries.

To search for movie schemas on the web, we picked a few keywords from current movie schemas and submitted them to Google. The HTML pages returned by Google often contain structured information that is like a schema. For example, below is a segment from the web page at http://www.moviepublicity.com/ppvvod/.

```
Director:Jim Isaac
Starring:Kane Hodder, Lexa Doig
Rating:R
Genre:Action/Horror
Run Time:91 minutes 30 seconds
Box Office: \$12,610,731
```

We gathered 17 movie schemas from the web in this way. For each schema we constructed a big query that involved all concepts in the schema. Note that some concepts like "Box Office" are not present in the 7 schemas. Thus we define "recall" on subqueries rather than on the whole big query. We define "recall" as the percentage of found translations out of all possible translations of subqueries. We define "precision" as the percentage of correct translations out of all translations of subqueries. The experiment showed a recall of 64% for all subquries. For the subquries semantically on the properties we have aligned (those in Table 2), it showed a recall of 72%. This result is already promising given that we've only aligned two pairs of schemas out of only 7 schemas. The experiment showed a precision of 100%. We attributed the high precision to the small knowledge base and the semantic correctness of the alignments. Given such a knowledge base, the algorithm either translates a subquery correctly or rejects it. We envision that with the size of knowledge base grows, the precision will decline and

SOME CLASSES IN DIFFERENT SCHEMAS

S1	[Movie]	[ThrillerMovie](subclass of [Movie])	[ComedyMovie](subclass of [Movie])
S2	[Movie]		
S3	[VideoLibraryItem]		
S4	[Movie]	[ThrillerMovie](subclass of [Movie])	[ComedyMovie](subclass of [Movie])
S5	[Movie]	[Movie]→movieGenre="Thriller"	[Movie]→movieGenre="Comedy"
S6	[Movie]	[Movie]→movieGenre→[Genre] →GenreName="Thriller"	[Movie] →movieGenre→[Genre] →GenreName="Comedy"
S7	[MovieInfo]		

SOME PROPERTIES IN DIFFERENT SCHEMAS

S1	movieName	directorName	actor
S2	Name	hasCrew→[Director]→PersonName	
S3	Title	Director	Actor
S4	movieName	movieDirector	leadingActor→[Artist]→artistName
S5	movieName	movieDirector→[Person]→personName	movieActor→[Person]→personName
S6	Title	Director	MainActor
S7	movieName	movieDirector	

S1	actress	rating→[Rating]→ratingType	
S2			genre
S3		Rating	
S4	leadingActress→[Artist]→artistName	movieRating	
S5		movieMPAARating	movieGenre
S6	MainActress	MPAA	movieGenre→[Genre]→GenreName
S7			

Table 2: Seven Movie Schemas

Step#	Rewriting Result	Rewriting Operation	Description
0	(?1,name,"The Ring")		Original Triple Pattern
1	(?1,S1:movieName,"The Ring")	localname matches URI	(?1,name,?2)→(?1,S1:movieName,?2)
2	(?1,S6:Title,"The Ring")	Property Alignment	(?1,S1:movieName,?2)↔(?1,S6:Title,?2)
3	...	...	...

Table 3: A Sample Series of Query Rewriting Steps

	Pruning Step 1
(?1,name,"The Ring")	(?1,S6:Title,"The Ring") (?1,S5:personName,"The Ring") (?1,S5:movieName,"The Ring") .....
(?1,moviedirector,?3)	~~(?1,S4:movieDirector,?3)(?3,S5:personName,??3)~~ ~~(?1,S3:Director,?3)(?3,S2:Name,??3)~~ (?1,S6:Director,?3)(?3,S6:Title,??3) (?1,S6:Director,?3) .....
(?1,type,ThrillerMovie)	~~(?1,type,S6:Movie)(?1,S5:movieGenre,?2)(?2,S6:GenreName,"Thriller")~~ ~~(?1,type,S6:Movie)(?1,S2:genre,?2)(?2,S6:GenreName,"Thriller")~~ (?1,type,S6:Movie)(?1,S6:movieGenre,?2)(?2,S6:GenreName,"Thriller") (?1,type,S5:Movie)(?1,S5:movieGenre,"Thriller") (?1,type,S4:ThrillerMovie) .....

Table 4: Pruning Step 1: Remove Rewritings Containing Different Schemas

	Pruning Step 2 & Checking Step
(?1,name,"The Ring") (?1,moviedirector,?3) (?1,type,ThrillerMovie)	(?1,S3:Title,"The Ring") (?1,S3:Director,?3) ~~()~~-
(?1,name,"The Ring") (?1,moviedirector,?3) (?1,type,ThrillerMovie)	(?1,S6:Title,"The Ring") (?1,S6:Director,?3) (?1,type,S6:Movie)(?1,S6:movieGenre,?2)(?2,S6:GenreName,"Thriller")
(?1,name,"The Ring") (?1,moviedirector,?3) (?1,type,ThrillerMovie)	(?1,S6:Title,"The Ring") (?1~~,S6:Director,?3)(?3,S6:Title,??3~~) (?1,type,S6:Movie)(?1,S6:movieGenre,?2)(?2,S6:GenreName,"Thriller")
(?1,name,"The Ring") (?1,moviedirector,?3) (?1,type,ThrillerMovie)	(?1~~,S5:personName,"The Ring~~") (?~~1,S5:movieDirector,?3~~) (?1,type,S5:Movie)(?1,S5:movieGenre,"Thriller")
.....	.....

Table 5: Pruning Step 2 and Checking Step: Pick Out Good Translations

the recall will increase. Note that our system does not guarantee 100% precision because we use a lot of guessing and aligning. The alignment between two terms seldom means these two terms are 100% equivalent.

5. APPLICATIONS AND FUTURE WORK

In this section, we discuss a couple of potential applications and extensions of our technique.

Information Search on the Semantic Web: The most natural application of our naive query translation technique will be to help information search on the Semantic Web. On the Semantic Web, we envision that numerous small schemas, rather than a few big schemas everyone must follow, will be created by people for their information management tasks. It is almost impossible to align all the schemas. It is also impossible for an information seeker to write all the different queries for different schemas. Thus it is important that, with a few alignments between a few schemas in the same domain, a naive user query can be translated into these schemas as well as others in the domain. The technique proposed in this paper represents our effort toward this great challenge.

In another project called "WebScripter" [19], we are developing a collaborative semantic annotation(CSA) tool for ordinary users to create metadata, and an easy-to-use report authoring tool for users to publish metadata as a user-friendly report as well as to align metadata from different schemas. With the grass-roots ontologies created by ordinary users using CSA and grass-roots alignment obtained from WebScripter, we hope that the technique we described in this paper would facilitate information sharing among WebScripter users.

Building Naive User Query Interface to an Existing Data Source: If we regard a naive user query as a mini-schema defined on-the-fly, it might be possible that, after accumulating and aligning a number of user queries, a system could interpret future naive user queries. Over time the system learns more rewriting rules and more synonyms, so it can produce increasingly robust and comprehensive response to new queries. A such-enhanced data source would provide a friendlier interface to information agents on the web.

Alignment-Carrying Information Agent: Other than enhancing a data source with a knowledge base of possible naive queries, if we arm an information agent with some alignment between some schemas of the domain of interest, will the agent be able to recognize future schemas it sees in the same domain, or at least, rewrite its task(a query) into those of the schemas? This kind of knowledgeable agent, or alignment-carrying agent, is likely to be more autonomous in information retrieval.

In a summary, there are interesting applications of the techniques we have developed. Exploring these applications, meanwhile testing the technique and discovering its deficiencies are our plan for the future work.

6. RELATED WORK

The research most related to our work is schema matching [17]. If we regard a naive user query as a mini-schema defined on-the-fly, translating naive user queries can be viewed as a special schema matching problem. However, there're significant differences between our work and schema matching, in the problem to be solved and in the techniques used.

Data schemas tend to be larger, more complex and rather static. Schema matching tries to make use of any helpful information such as name similarity, structure proximity [15], learning from data instances [9]. To further improve mapping accuracy, integration of all kinds of techniques into a single system is also used [8] [14]. Consequently, schema matching techniques are usually complex and time-consuming, and the matching process is generally assumed to happen offline. In contrast, naive user queries are normally short and dynamic. Timely response is also required.

Our translation approach makes use of a knowledge base of previous alignments. This distinguishes our work from many schema matching algorithms [15][9] [14] that only consider the two schemas at hand. The idea of reusing previous alignments is stated in [17] and further developed in [8]. However, [8] is not as flexible as our approach. In order to match $S1$ and $S2$ it requires the existence of S that has been already matched with $S1$ and $S2$, which makes it unusable for schemas unseen before. Alon Halevy [10] recently proposed to use a corpus of schemas to help schema matching. He also talked about the possibility of using such corpus to enable queries in users' own terminology. Our work goes one step further to also consider queries in users' own semantic structures. Furthermore, our idea of reusing previous alignments of *user queries* to translate future queries has not been seen in other's work.

There has been recent work [12] [4] [7] on enabling keyword-based search over relational databases. These systems try to compute a join of different table tuples which contains all the input keywords. Contrary to naive user queries, the relations between keywords are unclear, and it is difficult to specify what information users are looking for. As a result, users need to check that the relations between keywords in the returned tuples match user intents (For a set of keywords, there can be different join chains). Another human check is then required to extract the information users want from the tuples. Thus keyword-based search is more appropriate as a human activity rather than part of an automated computer program.

Natural language interface to databases provides another kind of query interface. Despite recent progress [16] [20], these systems remain difficult to implement. Natural language queries and SQL queries are two extremes and naive user queries are in the middle. We believe it is worthwhile to investigate whether a naive user query interface is easier to develop and performs better in terms of precision and recall.

Our query rewriting approach resembles a lot to those used in information integration systems [13] [5]. An information integration system translates a query between its global schema and local schemas. It assumes the existence of alignments between global schema and all local schemas. The query must be in one of the known schemas. In contrast, our work is on translating naive user queries that might not conform to any known schema. Our work is complementary to information integration in this sense.

7. CONCLUSION

The contributions of this paper are two-fold. First, we identified translating naive user queries as an important research problem. Translating naive queries is the process of translating queries in users' own terms and structures into those interpretable by data sources. Second, we proposed an approach to this problem. The approach utilizes schemas of different data sources and partial alignments between them to rewrite naive user queries into data-source-interpretable form. The approach is efficient and preliminary results were encouraging. We then showed how our technique could be an important component on the future Semantic Web. We also discussed possible applications of our technique, such as constructing naive-user-query translating interface to data sources and building Alignment-Carrying Information Agents on the current Web.

8. ACKNOWLEDGMENTS

The authors of the paper are supported by DARPA DAML program funding for WebScripter under contract number F30602-00-2-0576. We thank David Wilczynsky and Ellis Horowitz for providing their students' RDF Schemas. We thank Martin Frank for helpful comments.

9. REFERENCES

[1] Rdf vocabulary description language 1.0: Rdf schema. http://www.w3.org/TR/rdf-schema/.

[2] Rdql - rdf data query language. http://www.hpl.hp.com/semweb/rdql.htm.

[3] Resource description framework (rdf) model and syntax specification. http://www.w3.org/TR/1999/REC-rdf-syntax-19990222/.

[4] S. Agrawal and S. C. G. Das. Dbxplorer: A system for keyword-based search over relational databases. In *the 18th International Conference on Data Engineering (ICDE.02)*, 2002.

[5] Y. Arens, C. A. Knoblock, and W.-M. Shen. Query reformulation for dynamic information integration. *Journal of Intelligent Information Systems - Special Issue on Intelligent Information Integration*, 6(2/3):99–130, 1996.

[6] T. Berners-Lee, J. Hendler, and O. Lassila. The semantic web. *Scientific American*, May 2001.

[7] G. Bhalotia, A. Hulgeri, C. Nakhe, and S. Chakrabarti. Keyword searching and browsing in databases using banks. In *the 18th International Conference on Data Engineering (ICDE.02)*, 2002.

[8] H. Do and E. Rahm. Coma - a system for flexible combination of schema matching approaches, 2002.

[9] A. Doan, J. Madhavan, P. Domingos, and A. Halevy. Learning to map ontologies on the semantic web. In *The Eleventh International World Wide Web Conference*, 2002.

[10] A. Halevy, O. E. A. Doan, Z. Ives, and J. Madhavan. Crossing the structure chasm. In *the First Biennial Conference on Innovative Data Systems Research (CIDR)*, 2003.

[11] G. D. Hall, P. Approximate string matching. *Computing Survey*, 12(4):381–402, 1980.

[12] V. Hristidis and Y. Papakonstantinou. Discover: Keyword search in relational databases. In *the 28th VLDB Conference*, 2002.

[13] A. Y. Levy. Logic-based techniques in data integration. In J. Minker, editor, *Workshop on Logic-Based Artificial Intelligence, Washington, DC, June 14–16, 1999*, College Park, Maryland, 1999. Computer Science Department, University of Maryland.

[14] J. Madhavan, P. A. Bernstein, and E. Rahm. Generic schema matching with cupid. In *The VLDB Journal*, pages 49–58, 2001.

[15] S. Melnik, H. Molina-Garcia, and E. Rahm. Similarity flooding: A versatile graph matching algorithm. In *the International Conference on Data Engineering (ICDE)*, 2002.

[16] A. Popescu, O. Etzioni, and H. Kautz. Towards a theory of natural language interfaces to databases. In *IUI*, 2003.

[17] E. Rahm and P. Bernstein. On matching schemas automatically. Technical report, Microsoft Research, Redmon, WA, 2001. MSR-TR-2001-17.

[18] R. Tarjan. Fast algorithms for solving path problems. *Journal of the ACM*, 3(28):591–642, 1981.

[19] B. Yan, M. Frank, P. Szekely, R. Neches, and J. Lopez. Webscripter: Grass-roots ontology alignment via end-user report authoring. In *the Second International Semantic Web Conference*, Octor 2003.

[20] J. M. Zelle and R. J. Mooney. Learning to parse database queries using inductive logic programming. In *the Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, 1996.