Generic Querying of Relational Databases using Natural Language Generation Techniques

Catalina Hallett
Center for Research in Computing
The Open University
Walton Hall, Milton Keynes
United Kingdom
c.hallett@open.ac.uk

Abstract

This paper presents a method of querying databases by means of a natural language-like interface which offers the advantage of minimal configuration necessary for porting the system. The method allows us to first automatically infer the set of possible queries that can apply to a given database, automatically generate a lexicon and grammar rules for expressing these queries, and then provide users with an interface that allows them to pose these queries in natural language without the well-known limitations of most natural language interfaces to databases. The way the queries are inferred and constructed means that semantic translation is performed with perfect reliability.

1 Introduction

Natural Language interfaces to databases (hereafter NLIDBs) have long held an appeal to both the databases and NLP communities. However, difficulties associated with text processing, semantic encoding, translation to database querying languages and, above all, portability, have meant that, despite recent advances in the field, NLIDBs are still more a research topic than a commercial solution.

Broadly, research in NLIDBs has focused on addressing the following fundamental, interdependent issues[1]:

- domain knowledge aquisition (Frank et al., 2005);
- interpretation of the input query, including parsing and semantic disambiguation, semantic interpretation and transformation of the query to an intermediary logical form (Hendrix et al., 1978; Zhang et al., 1999; Tang and Mooney, 2001; Popescu et al., 2003; Kate et al., 2005);
- translation to a database query language (Lowden et al., 1991; Androutsopoulos, 1992);
- portability (Templeton and Burger, 1983; Kaplan, 1984; Hafner and Godden, 1985; Androutsopoulos et al., 1993; Popescu et al., 2003)

In order to recover from errors in any either of these steps, most advanced NLIDB systems will also incorporate some sort of cooperative user feedback module that will inform the user of the inability of the system to construct their query and ask for clarification.

We report here on a generic method we have developed to automatically infer the set of possible queries that can apply to a given database, and an interface that allows users to pose these questions in natural language but without the previously mentioned drawbacks of most NLIDBs . Our work is substantially different from previous research in that it does not require the user to input free text queries, but it assists the user in composing query through a natural language-like interface. Consequently, the necessity for syntactic parsing and semantic interpretation is eliminated. Also, since users are in control of the meaning of the query they compose, ambiguity is not an issue.

Our work builds primarily on two directions of research: conceptual authoring of queries via

[1]The extent of NLIDB research is such that it is beyond the scope of this paper to reference a comprehensive list of projects in this area. For reviews on various NLIDBs , the reader is referred to (Androutsopoulos et al., 1995).

Proceedings of the Fourth International Natural Language Generation Conference, pages 95–102,
Sydney, July 2006.

WYSIWYM interfaces, as described in section 2, and NLIDB portability research. From the perspective of the query composing technique, our system resembles early menu-based techniques, such as Mueckstein (1985), NL-Menu (Tennant et al., 1983) and its more recent re-development Lingo-Logic (Thompson et al., 2005). This resemblance is however only superficial. Our query editing interface employs natural language generation techniques for rendering queries in fluent language; it also allows the editing of the semantic content of a query rather than its surface form, which allows seamless translation to SQL .

As in (Zhang et al., 1999), our system makes use of a semantic graph as a mean of representing the database model. However, whilst Zhang et al (1999) use the Semantic Graph as a resource for providing and interpreting keywords in the input query, we use this information as the main means of automatically generating query frames.

2 WYSIWYM interfaces for database querying

Conceptual authoring through WYSIWYM editing alleviates the need for expensive syntactic and semantic processing of the queries by providing the users with an interface for editing the conceptual meaning of a query instead of the surface text (Power and Scott, 1998).

The WYSIWYM interface presents the contents of a knowledge base to the user in the form of a natural language feedback text. In the case of query editing, the content of the knowledge base is a yet to be completed formal representation of the users query. The interface presents the user with a natural language text that corresponds to the incomplete query and guides them towards editing a semantically consistent and complete query. In this way, the users are able to control the interpretation that the system gives to their queries. The user starts by editing a basic query frame, where concepts to be instantiated (anchors) are clickable spans of text with associated pop-up menus containing options for expanding the query.

Previously, WYSIWYM interfaces have proved valid solutions to querying databases of legal documents and medical records (Piwek et al., 2000), (Piwek, 2002), (Hallett et al., 2005).

As a query-formulation method, WYSIWYM provides most of the advantages of NLIDBs , but overcomes the problems associated with natural language interpretation and of users attempting to pose questions that are beyond the capability of the system or, conversely, refraining from asking useful questions that are in fact within the system's capability. However, one of the disadvantages of the WYSIWYM method is the fact that domain knowledge has to be manually encoded. In order to construct a querying interface for a new database, one has to analyse the database and manually model the queries that can be posed, then implement grammar rules for the construction of these queries. Also, the process of transforming WYSIWYM queries into SQL or another database querying language has previously been database-specific. These issues have made it expensive to port the interface to new databases and new domains.

The research reported here addresses both these shortcomings by providing a way of automatically inferring the type of possible queries from a graph representation of the database model and by developing a generic way of translating internal representations of WYSIWYM constructed queries into SQL .

3 Current approach

In the rest of the paper, we will use the following terms: a *query frame* refers to a system-generated query that has not been yet edited by the user, therefore containing only unfilled WYSIWYM anchors. An *anchor* is part of the WYSIWYM terminology and means a span of text in a partially formulated query, that can be edited by the user to expand a concept. Anchors are displayed in square brackets (see examples in section 3.3).

To exemplify the system behaviour, we will use as a case study the MEDIGAP database, which is a freely downloadable repository of information concerning medical insurance companies in the United States. We have chosen this particular database because it contains a relatively wide range of entity and relation types and can yield a large number of types of queries. In practice we have often noticed that large databases tend to be far less complex.

3.1 System architecture

Figure 1 shows the architecture of the querying system. It receives as input a model of the database semantics (the semantic graph) and it automatically generates some of the compo-

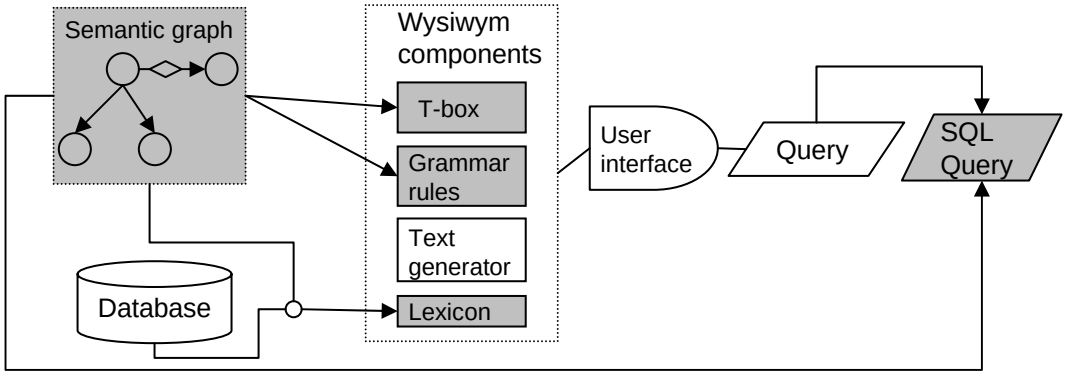

Figure 1: System architecture

nents and resources (highlighted in grey) that in previous WYSIWYM querying systems were constructed manually. Finally, it implements a module that translates the user-composed query into SQL .

The components highlighted in grey are those that are constructed by the current system.

The **T-box** describes the high-level components of the queries. It is represented in Profit notation (Erbach, 1995) and describes the composition of the query frames (the elements that contribute to a query and their type) . A fragment of the semantic graph displayed in 2 will generate the following fragment of t-box:

```
query > [about_company, about_state,
about_phone, about_ext].
about_company > [company_state, company_phone,
company_ext].
company_state intro [company:company_desc].
company_desc intro [comp:comp_desc, phone:phone_desc,
ext:ext_desc].
state_desc > external('dbo_vwOrgsByState_StateName').
comp_desc > external('dbo_vwOrgsByState_org_name').
phone_desc > external('dbo_vwOrgsByState_org_phone').
ext_desc > external('dbo_vwOrgsByState_org_ext').
```

The **grammar rules** are also expressed in Profit, and they describe the query formulation procedure. For example, the following rule will be generated automatically to represent the construction procedure for the query in Example (1.1):

```
rule(english, company_state,
   meaning!(<description &
            predicate!company_state &
            properties![attribute!comp & value!Comp]) &
   layout!level!question &
   cset![meaning!in &
         syntax!category!prep &
         layout!level!word,

         meaning!which &
         syntax!category!int &
         layout!level!word,

         meaning!state &
         syntax!category!np &
         layout!level!word,

         meaning!be &
         syntax!(category!vb & form!pres),
         layout!level!word,

         meaning!Comp &
         syntax!category!np &
         layout!level!phrase,

         meaning!locate &
         syntax!(category!vb & form!part),
         layout!level!word]).
```

In addition to the grammar rules automatically generated by the system, the WYSIWYM package also contains a set of English grammar rules (for example, rules for the construction of definite noun phrases or attachment of prepositional phrases). These rules are domain independent, and therefore a constant resource for the system.

The **lexicon** consists of a list of concepts together with their lexical form and syntactic category. For example, the lexicon entry for *insurance company* will look like:

```
word(english, meaning!company &
syntax!(category!noun & form!name) &
cset!'insurance company')).
```

3.2 Semantic graph

The semantics of a relational database is specified as a directed graph where the nodes represent elements and the edges represent relations between elements. Each table in the database can be seen as a subgraph, with edges between subgraphs representing a special type of join relation.

Each node has to be described in terms of its semantics and, at least for the present, in terms of its linguistic realisation. The semantic type of a node is restricted by the data type of the corresponding entity in the database. A database entity of type *String* can belong to one of the following semantic categories: *person, organization, location (town, country), address (street or complete address), telephone number, other name, other object*. Similarly, numerical entities can have the semantic type: *age, time (year, month, hour), length,*

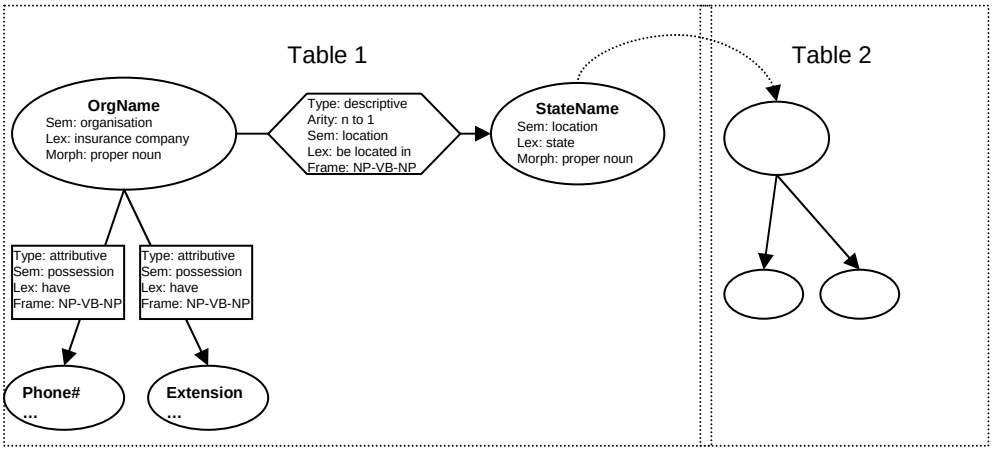

Figure 2: Example of a semantic graph

weight, value, height. The data type *date* has only one possible semantic type, which is *date*. These semantic types have proved sufficient in our experiments, however this list can be expanded if necessary.

Apart from the semantic type, each node must specify the linguistic form used to express that node in a query. For example, in our case study, the field *StateName* will be realised as *state*, with the semantic category *location*. Additionally, each node will contain the name of the table it belongs to and the name of the column it describes.

Relations in the semantic graph are also described in terms of their semantic type. Since relations are always realised as verbs, their semantic type also defines the subcategorisation frame associated with that verb. For the moment, subcategorisation frames are imported from a locally compiled dictionary of 50 frequent verbs. The user only needs to specify the semantic type of the verb and, optionally, the verb to use. The system automatically retrieves from the dictionary the appropriate subcategorisation frame. The dictionary has the disadvantage of being rather restricted in coverage, however it alleviates the need for the user to enter subcategorisation frames manually, a task which may prove tedious for a user without the necessary linguistic knowledge. However, we allow users to enter new frames in the dictionary, should their verb or category of choice not be present. A relation must also specify its arity.

This model of the database semantics is partially constructed automatically by extracting database metadata information such as data types and value ranges and foreign keys. The manual effort involved in creating the semantic graph is reduced to the input of semantic and linguistic information.

3.3 Constructing queries

We focus our attention in this paper to the construction of the most difficult type of queries: complex wh-queries over multiple database tables and containing logical operators. The only restriction on the range of wh-queries we currently construct is that we omit queries that require inferences over numerical and date types.

Each node in the semantic graph can be used to generate a number of query frames equal to the number of nodes it is connected to in the graph. Each query frame is constructed by pairing the current node with each other of the nodes it is linked to. By *generation of query frames* we designate the process of automatically generating Profit code for the grammar rule or set of rules used by WYSIWYM , together with the T-box entries required by that particular rule.

If we consider the graph presented in Fig.2, and focus on the node *orgName*, the system will construct the query frames:

```
Example (1):
1.  In which state is [some insurance
company] located?
2.  What phone number does [some
insurance company] have?
3.  What extension does [some insurance
company] have?
```

If we consider the first query in the example above, the user can further specify details about

the company by selecting the *[some insurance company]* anchor and choosing one of the options available (which themselves are automatically generated from the database in question). This information may come from one or more tables. For example, one table in our database contains information about the insurance companies contact details, whilst another describes the services provided by the insurance companies. Therefore, the user can choose between three options: *contact details*, *services* and *all*. Each selection triggers a text regeneration process, where the feedback text is transformed to reflect the user selection, as in the example below:

```
Example (2):
1.  In which state is [some insurance
company] that has [some phone number]
and [some extension] located?
2.  In which state is [some insurance
company] that offers [some medical
insurance plan] and [is available] to
people over 65 located?
3.  In which state is the insurance
company with the following features
located:
• It has [some phone number] and [some
extension]
and
• It offers [some medical insurance
plan] and [is available] to people over
65
```

Figure 3 shows a snapshot of the query editing interface where query (2.1) is being composed.

Each query frame is syntactically realised by using specially designed grammar rules. The generation of high level queries such as those in Example (1.1) relies on basic query syntax rules. The semantic type of each linked element determines the type of wh-question that will be constructed. For example, if the element has the semantic type *location*, we will construct *where* questions, whilst a node with the semantic type *PERSON* will give rise to a *who*-question. In order to avoid ambiguities, we impose further restrictions on the realisation of the query frames. If there is more than one *location*-type element linked to a node, the system will not generate two *where* query frames, which would be ambiguous, but more specific *which* queries. For example, our database contains two nodes of semantic type *location* linked to the node *OrgName*. The first describes the state where an insurance company is located, the second its address. The query frames generated will be:

```
Example (3):
1.  In which states is some insurance
company located?
2.  At what addresses is some insurance
company located?
```

The basic grammar rule pattern for queries based on one table only states that elements linked to a particular node will be realised in relative clauses modifying that node. For example, in Example (2.1), the nodes *phones* and *ext* are accessible from the node *orgName*, therefore will be realised in a relative clause that modifies *insurance company*.

In the case where the information comes from more than one table, it is necessary to introduce more complex layout features in order to make the query readable. For each table that provides information about the focused element we generate bulleted lines as in Example (2.3).

Each question frame consists of a bound element[2], i.e., the user cannot edit any values for that particular element. This corresponds to the information that represents the answer to the questions. In example (2), the bound element is *state*. All other nodes will be realised in the feedback text as anchors, that are editable by users. One exception is represented by nodes that correspond to database elements of boolean type. In this case, the anchor will not be associated to a node, but to a relation, as in Example (2.3) (the availability of an insurance plan is a boolean value). This is to allow the verb to take negative form - in our example, one can have *is available to people over 65* or *is not available to people over 65*.

Since not all anchors have to be filled in, one query frame can in fact represent more than one real-life question. In example (4), one can edit the query to compose any of the following corresponding natural language questions:

```
Example (4):
1.  In which state is the insurance
company with the phone number 8008474836
located?
2.  In which state is the insurance
```

[2]In fact, a single element can be replaced of any number of elements of the same type linked by conjunctions or disjunctions. However, we will refer to a single element by way of simplification. The process of inferring queries remains esentially the same.

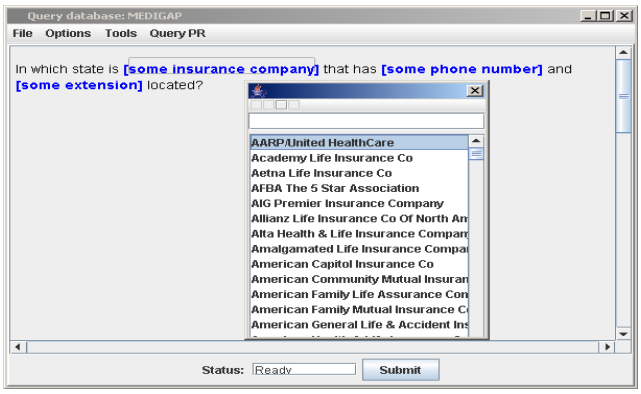

Figure 3: Query editing interface snapshot

company Thrivent Financial for Lutherans with the phone number 8008474836 located?

3. *In which state is the insurance company Thrivent Financial for Lutherans with the phone number 8008474836 and extension 8469 located?*

The actual values of anchors are extracted from the database and transformed into correct lexicon entries on a per-need basis. The strings associated with a value (e.g. *Thrivent Financial for Lutherans*) are retrieved from the database table and column indicated in the description of the node that was used for generating the anchor (e.g. *orgName*) and the syntactic role (e.g. *proper noun*) is given by the syntactic information associated with the node.

3.4 Query translation module

Once a query has been constructed, it is represented internally as a directed acyclic graph. Moreover, each node in the graph can be mapped into a node in the semantic graph of the database. The translation module transforms a contructed query to an SQL statement by parsing the query graph and combining it with the corresponding elements in the semantic graph.

The SELECT portion of the statement contains the focused element. The WHERE portion contains those nodes in the question graph that correspond to edited anchors. For constructing the FROM portion of the statement, we extract, from the semantic graph, for each SELECTED element information about their corresponding database table.

For example, if we assume that in Example (2.1) the user has specified the name of the company and its phone number, the SQL statement generated will be:

```
SELECT dbo_vwOrgsByState.StateName
FROM dbo_vwOrgsByState
WHERE org_name="Thrivent Financial for
                Lutherans"
  And org_phone="8008474836";
```

4 Evaluation

4.1 Usability

A recent study of the usability of a WYSIWYM type of interface for querying databases (Hallett et al., 2006) has shown that users can learn how to use the interface after a very brief training and succeed in composing queries of quite a high level of complexity. They achieve near-perfect query construction after the first query they compose. The study also showed that the queries as they appear in the WYSIWYM feedback text are unambiguous — not only to the back-end system — but also to the user, i.e., users are not misled into constructing queries that may have a different meaning than the one intended. Additionally, it appears that expert users of SQL , with expert knowledge of the underlying database, find the query interface easier to use than querying the database directly in SQL . We consider that most of the conclusions drawn in (Hallett et al., 2006) apply to the current system. The only difference may appear in assessing the ambiguity of the feedback text. Since the query construction rules used for our system are generated automatically, it is likely that the feedback text may be less fluent and, potentially, more ambiguous than a feedback text generated using manually constructed rules, as in (Hallett et al., 2006). We have not yet addressed this issue in a formal evaluation of the current system.

4.2 Coverage

We have assessed the coverage of the system using as our test set a set of English questions posed over a database of geographical information GEOBASE, as in (Tang and Mooney, 2001) and (Popescu et al., 2003). Our first step was to convert the original Prolog database (containing about 800 facts) into a relational database. Then we tested how many of the 250 human produced questions in the test set can be constructed using our system.

There are several issues in using this particular dataset for testing. Since we do not provide a pure natural language interface, the queries our system can construct are not necessarily expressed in the same way or using the same words as the questions in the test set. For example, the question *"How high is Mount McKinley?"* in the test set is equivalent to *"What is the height of Mount McKinley?"* produced by our system. Similarly, *"Name all the rivers in Colorado."* is equivalent to *"Which rivers flow through Colorado?"*. Also, since the above test set was designed for testing and evaluating natural language interfaces, many of the questions have equivalent semantic content. For example, *"How many people live in California?"* is semantically equivalent to *"What is the population of California?"*. Similarly, there is no difference in composing and analysing *"What is the population of Utah?"* and *"What is the population of New York City?"*.

Out of 250 test questions, 100 had duplicate semantic content and the remaining 150 had original content. On the whole test set of 250 questions, our system was able to generate query frames that allow the construction of 145 questions, therefore 58%. The remaining 42% of questions belong to a single type of questions that our current implementation cannot handle, which is questions that require inferences over numerical types, such as *Which is the highest point in Alaska?* or *What is the combined area of all 50 states?*.

Similar results are achieved when testing the system on the 150 relevant questions only: 60% of the questions can be formulated, while the remaining 40% cannot.

4.3 Correctness

The correctness of the SQL generated queries was assessed on the subset of queries that our system can formulate out of the total number of queries in the test set. We found that the correct SQL was produced for all the generated WYSIWYM queries produced.

5 Conclusions & Further work

Our method presents three main advantages over other natural language interfaces to databases:
1. It is easily customizable for new domains and databases.
2. It eliminates errors in parsing and query-to-SQL translation.
3. It makes clear to the user the full range of possible queries that can be posed to any given database.

From a user's point of view, one could argue that our method is less natural to use than one that allows unconstrained (or less constrained) natural language input. It could also be said that while syntactically correct, the queries as presented to the user may not be as fluent as human-authored questions. These possible disadvantages are, in our opinion, outweighed by the clarity of the query composition process, since the user is fully in control of the *semantic content* of the query she composes; they are unambiguous to both the user and the back-end system.

We are currently extending this work to cover more complex queries that require inferences and ones that contain elements linked through temporal relations. We will also refine the query layout procedures to allow complex queries to be presented in a more intuitive way. Additionally, we are about to begin work on automating the the construction of the semantic graph. We expect that some of the semantic and syntactic information that, at the moment, has to be manually entered in the description of the semantic graph can be inferred automatically from the database content.

Acknowledgement

The work described in this paper is part of the Clinical E-Science Framework (CLEF) project, funded by the Medical Research Council grant G0100852 under the E-Science Initiative. We gratefully acknowledge the contribution of our clinical collaborators at the Royal Marsden and Royal Free hospitals, colleagues at the National Cancer Research Institute (NCRI) and NTRAC and to the CLEF industrial collaborators.

References

I. Androutsopoulos, G.Ritchie, and P.Thanitsch. 1993. An effcient and portable natural language query interface for relational databases. In *Proceedings of the 6th International Conference on Industrial Engineering Applications of Artificial Intelligence and Expert Systems Edinburgh*, pages 327–330.

I. Androutsopoulos, G.D. Ritchie, and P.Thanisch. 1995. Natural language interfaces to databases - an introduction. *Natural Language Engineering*, 2(1):29–81.

I. Androutsopoulos. 1992. Interfacing a natural language front end to a relational database. Master's thesis, Department of Artificial Intelligence University of Edinburgh.

Gregor Erbach. 1995. ProFIT – prolog with features, inheritance, and templates. In *Proceedings of the 7th Conference of the European Chapter of the Asscociation for Computational Linguistics, EACL-95*, Dublin, Ireland.

A. Frank, Hans-Ulrich Krieger, Feiyu Xu, Hans Uszkoreit, Berthold Crysmann, Brigitte Jorg, and Ulrich Schafer. 2005. Querying structured knowledge sources. In *AAAI-05 Workshop on Question Answering in Restricted Domains*, Pittsburgh, Pennsylvania.

Carole D. Hafner and Kurt Godden. 1985. Portability of syntax and semantics in datalog. *ACM Trans. Inf. Syst.*, 3(2):141–164.

C. Hallett, D. Scott, and R.Power. 2005. Intuitive querying of ehealth data repositories. In *Proceedings of the UK E-Science All-Hands Meeting*, Nottingham, UK.

C. Hallett, D. Scott, and R.Power. 2006. Evaluation of the clef query interface. Technical Report 2006/01, Department of Computing, The Open University.

Gary G. Hendrix, Earl D. Sacerdoti, Daniel Sagalowicz, and Jonathan Slocum. 1978. Developing a natural language interface to complex data. *ACM Trans. Database Syst.*, 3(2):105–147.

S. Jerrold Kaplan. 1984. Designing a portable natural language database query system. *ACM Trans. Database Syst.*, 9(1):1–19.

R.J. Kate, Y.W. Wong, and R.J. Mooney. 2005. Learning to transform natural to formal languages. In *Proceedings of the Twentieth National Conference on Artificial Intelligence (AAAI-05)*, pages 1062–1068, Pittsburgh, PA.

B.G.T. Lowden, B.R. Walls, A. De Roeck, C.J. Fox, and R. Turner. 1991. A formal approach to translating english into sql. In Jackson and Robinson, editors, *Proceedings of the 9th British National Conference on Databases*.

Eva-Martin Mueckstein. 1985. Controlled natural language interfaces (extended abstract): the best of three worlds. In *CSC '85: Proceedings of the 1985 ACM thirteenth annual conference on Computer Science*, pages 176–178, New York, NY, USA. ACM Press.

P. Piwek, R. Evans, L. Cahill, and N. Tipper. 2000. Natural language generation in the mile system. In *Proceedings of the IMPACTS in NLG Workshop*, Schloss Dagstuhl, Germany.

P. Piwek. 2002. Requirements definition, validation, verification and evaluation of the clime interface and language processing technology. Technical Report ITRI-02-03, ITRI, University of Brighton.

Ana-Maria Popescu, Oren Etzioni, and Henry Kautz. 2003. Towards a theory of natural language interfaces to databases. In *IUI '03: Proceedings of the 8th international conference on Intelligent user interfaces*, pages 149–157, New York, NY, USA. ACM Press.

Richard Power and Donia Scott. 1998. Multilingual authoring using feedback texts. In *Proceedings of 17th International Conference on Computational Linguistics and 36th Annual Meeting of the Association for Computational Linguistics (COLING-ACL 98)*, pages 1053–1059, Montreal, Canada.

Lappoon R. Tang and Raymond J. Mooney. 2001. Using multiple clause constructors in inductive logic programming for semantic parsing. In *EMCL '01: Proceedings of the 12th European Conference on Machine Learning*, pages 466–477, London, UK. Springer-Verlag.

Marjorie Templeton and John Burger. 1983. Problems in natural-language interface to dbms with examples from eufid. In *Proceedings of the first conference on Applied natural language processing*, pages 3–16, Morristown, NJ, USA. Association for Computational Linguistics.

Harry R. Tennant, Kenneth M. Ross, and Craig W. Thompson. 1983. Usable natural language interfaces through menu-based natural language understanding. In *CHI '83: Proceedings of the SIGCHI conference on Human Factors in Computing Systems*, pages 154–160, New York, NY, USA. ACM Press.

C. Thompson, P. Pazandak, and H. Tennant. 2005. Talk to your semantic web. *IEEE Internet Computing*, 9:75–78.

Guogen Zhang, Wesley W. Chu, Frank Meng, and Gladys Kong. 1999. Query formulation from high-level concepts for relational databases. In *UIDIS '99: Proceedings of the 1999 User Interfaces to Data Intensive Systems*, page 64, Washington, DC, USA. IEEE Computer Society.