

# A *STEP* Towards Realizing Codd's Vision of Rendezvous with the Casual User

Michael J. Minock  
Umeå University, Department of Computing Science  
SE-901 87 Umeå, Sweden  
mjm@cs.umu.se

## ABSTRACT

This demonstration showcases the STEP system for natural language access to relational databases. In STEP an administrator authors a highly structured semantic grammar through coupling phrasal patterns to elementary expressions within a decidable fragment of tuple relational calculus. The resulting phrasal lexicon serves as a bi-directional grammar, enabling the generation of natural language from tuple relational calculus and the inverse parsing of natural language to tuple calculus. This ability to both understand and generate natural language enables STEP to engage the user in clarification dialogs when the parse of their query is of questionable quality. The STEP system is nearing completion and will soon be field tested in several domains.

## 1. INTRODUCTION

The databases that back the 'deep web' are typically accessed through either forms based or navigational point and click interfaces. And while this is often sufficient for simple databases with very well circumscribed uses, these interfaces quickly become cumbersome for more complex domains and uses. In this paper we reopen the old question of whether natural language interfaces (NLIs) offer a way to overcome such difficulties. Historically such efforts have aroused great interest [4, 1, 5]. In fact Codd himself designed and partially implemented the RENDEZVOUS system in which users could access databases via relatively unrestricted natural language. In Codd's system, and in the system described here, special emphasis is placed on *query paraphrasing* and in engaging users in *clarification dialogs* when there is difficulty parsing user input.

The primary argument in favor of NLIs is that people already know natural language and therefore presumably require very little or no training to access relevant information. While it can also be argued that forms and navigation based interfaces require very little training, it must be noted that this is only so for rather simple databases (or views). Once the conceptual complexity (roughly the number of tables)

goes beyond a certain bound, these systems have difficulty supporting ad hoc queries from casual users. Natural language also has the special advantage that it is relatively easy to express negation (e.g., "projectors not built in Asia"), logical quantification (e.g., "distributors with every InFocus projector in stock") or superlative queries (e.g., "3 brightest projectors under \$2000 and over 3000 contrast"). In addition NLIs, in principle at least, enable *meta-level questions* about the structure and nomenclature of the database (e.g., "What does DLP mean?"). Finally NLIs give the capability for users to participate in dialogues where they build up a context and make anaphoric references to previous content (e.g., "What is *its* shipping time?").

Considering all of these desirable features of NLIs, why aren't they typically available on today's web? The answer to this question is not immediately forthcoming and requires a look back. Work began in the late 70's and continued through the 80's toward building NLIs to databases and by the end of the 80's two dominant approaches had emerged: the *semantic grammar* approach and the *transportable systems* approach. Semantic grammar approaches (e.g., [15]) build grammar rules based directly on the tables in the underlying database. Their advantage is that, given enough time, one can actually build fairly robust solutions for closed domains. A disadvantage is that building them is tedious and error prone. Furthermore the effort is not transportable to new database domains. Transportable systems (e.g., [6]) arose as a response to the tedious configuration requirements of semantic grammar based systems. The idea is to use large scale domain-independent grammars, developed in the linguistics community, to map user requests to intermediate *logical form*. Using *translation knowledge*, the logical form is then translated to a logical query expressed in the vocabulary of the relations of the actual database. Hence building an interface over a new database requires a set of domain-specific lexical entries and the specification of translation knowledge, but does not require new linguistic syntax rules to be defined. Sidestepping the fact that obtaining domain-specific lexicons and translation knowledge may be difficult, transportable approaches were trumpeted as superior to semantic grammars in the late 80's. Large scale grammar work in linguistics dovetailed nicely with the transportable approach, and plans were drawn up to fully develop and commercialize the results.

Had expectations been fulfilled, NLIs to databases would by now be in wide use. Clearly something went wrong.

Permission to copy without fee all or part of this material is granted provided that the copies are not made or distributed for direct commercial advantage, the VLDB copyright notice and the title of the publication and its date appear, and notice is given that copying is by permission of the Very Large Data Base Endowment. To copy otherwise, or to republish, to post on servers or to redistribute to lists, requires a fee and/or special permission from the publisher, ACM.

VLDB '07, September 23-28, 2007, Vienna, Austria.  
Copyright 2007 VLDB Endowment, ACM 978-1-59593-649-3/07/09.

## 1.1 Problems with NLI to Databases

From a human factors point of view, the primary problem with NLI to databases is that the linguistic and conceptual coverage of a system is not obvious. A typical pattern is that the user first overestimates the coverage offered by the system and then, after the system fails, the user revises their appraisal of what the system ‘knows about’ downward. The problem is that failure can be due to linguistic limitations, not just missing database content. Empirical work suggests that linguistic limitations are more serious because they can lead users to incorrectly assume that some content is not covered by the database when it in fact is [13]. In such cases the user’s understanding of the system becomes confused and they experience frustration trying to reconcile it.

Another factor that has limited the usefulness of such interfaces is their inability to handle informal register. One example that has been extensively documented is the tendency for ‘and’ to actually mean logical ‘or’ in the informal register. As in, “list the projectors built by InFocus and Mitsubishi,” not meaning the projectors manufactured by *both* InFocus and Mitsubishi.

Users, especially when interacting with a computer, prefer brevity over grammatical correctness. Thus in many cases their requests are simply noun phrases or their request contain anaphoric references or rely on context. These are very significant challenges, especially for transportable systems which are based on theoretical idealizations of language, not the kind of slop that real users use.

A final problem of particular relevance to the database community is the manifest inability of NLI to insure semantic correctness of user queries and operations. Although Codd advised the community to include an accurate paraphrase-and-verify step [4], it seems that developed systems seldom take this requirement seriously and instead simply translate the user’s query to SQL, applied it and then presented the answers, perhaps along with the SQL. This not only gives a user qualms about trusting the system, it also completely precludes the possibility of using NLI to conduct updates or other side-effecting operations.

For better or worse, the database and computational linguistics communities drifted away from NLI for databases. First, it was not clear how to overcome the above difficulties, especially if one was to adhere to the transportable approach. Second, linguistic efforts were more profitably applied toward developing large scale grammars and other resources that would become very useful in information retrieval. Third, modern WIMP (Windows, Icons Menus and Pointers) based interfaces were satisfying many, but not all, of the demands that NLI to databases were meant to satisfy. Finally, there were exciting opportunities, such as object-oriented databases that lured away the database talent. As time passed there were many more areas that grabbed the attention. In short, NLI to databases were largely abandoned as the database and computational linguistics community embraced other exciting possibilities.

## 2. OVERVIEW OF STEP

The system STEP [9, 11, 10] (Originally, Schema Tuple Expression Processor) takes a fresh look at this old problem. Unlike all other work in the area, STEP uses a bi-directional grammar to paraphrases logical queries back to the user in natural language. In other words, the user is al-

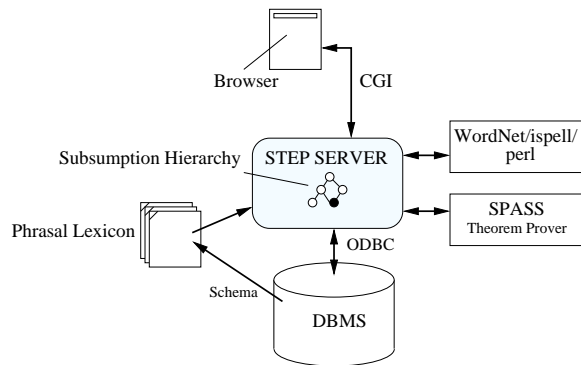


Figure 1: STEP system architecture

ways informed of how the system interpreted their question or request. Like Codd, we assume that systems that do not accurately paraphrase queries back to the user are, in practice, near worthless; *users must be able to verify the semantic correctness of their requests or operations*. In addition STEP also supports natural language updates to databases [11] and tools to efficiently integrate STEP with arbitrary databases. Administrators are assumed to understand primary and foreign keys, SQL and tuple calculus, but are not assumed to have a deep background in linguistics.

The primary linguistic knowledge that guides both query analysis as well as query paraphrasing is a *phrasal lexicon*, a highly structured semantic grammar. STEP exploits the decidability of the Schönfinkel-Bernays class [2] of first-order logic to decide query emptiness and containment. Such properties are fundamental to our representation and processing of linguistic and relational knowledge. Furthermore the availability of fast theorem provers (e.g., [16]) make such calculations feasible in real time.

Figure 1 shows the architecture of STEP, which is written in LISP and runs under CLISP on both the Linux and Windows platforms. It runs as a server that is called through CGI from a web browser and employs spell checkers, regular expression based extractors and WORDNET to preprocess user supplied strings. STEP issues satisfiability queries to the SPASS theorem prover [16] and relational queries to back-end databases via ODBC. At start-up time STEP compiles the phrasal lexicon into a subsumption hierarchy and materializes attribute value maps from the database. The system is then ready to accept browser-submitted requests, which it can typically answer in under a second.

Figure 2 shows an instance of the web-based interface for a projector database. Users enter their queries on the input field and obtain answers in the area immediately below. Note the sloppy user input and the automatically generated query paraphrase. The picture of the projector helps inform the user of the information in the database and the types of conditions and constants that questions may employ. Finally there is an ‘example queries’ button that can give the user a sample of the types of queries that may be asked through the interface.

## 3. THE PHRASAL LEXICON

The phrasal lexicon is the primary configuration structure that administrators must author to tie STEP to their domain database. In short, the administrator authors a set of

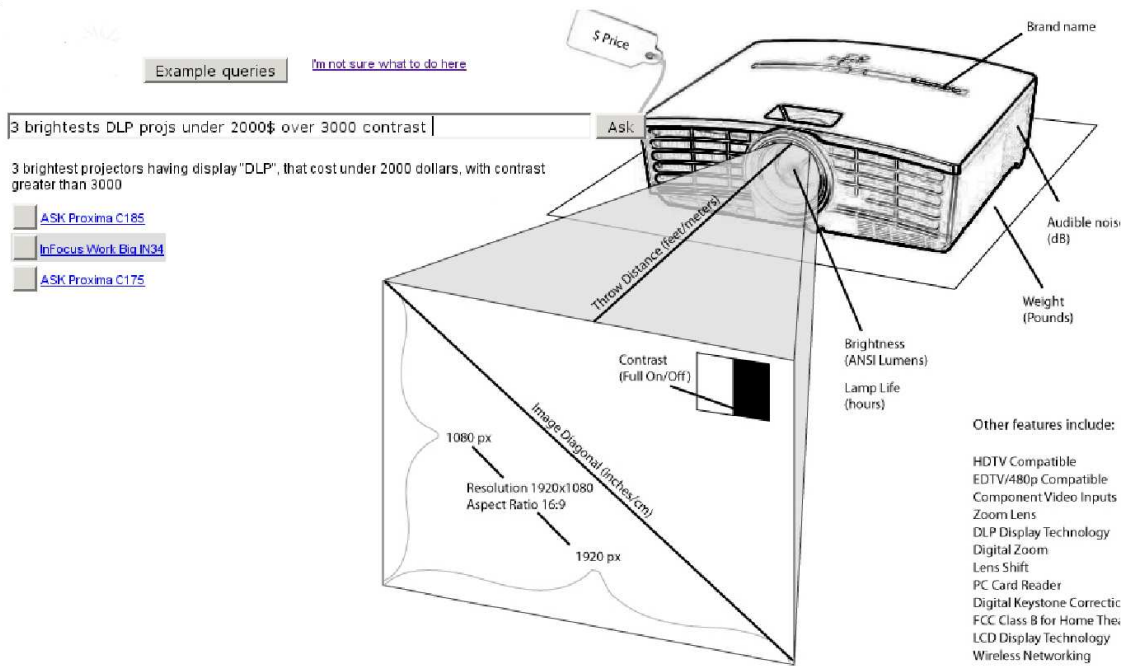


Figure 2: Querying a digital projector database

entries to cover the domain schema. Although the presentation in this paper will go into the details of these entries, we have developed a web-base authoring tool that shields the administrator from having to hand code such entries.

An entry associates a single elementary tuple calculus expression with a set of *patterns*, each pattern providing an alternative way to linguistically express the elementary expression. We begin with a very simple example entry:

$\{x|Projector(x) :$   
 “[ ] projectors [ ]”  
 “[ ] proj's [ ]”

In this entry the tuple calculus expression  $\{x|Projector(x)\}$  is coupled with two patterns. One the normal way to refer to a projector and the second an abbreviation that some users might prefer.

In the following entry we see a more complex elemental pattern that includes a *template parameter*, *c*.

$\{x|Projector(x) \wedge x.type = c\} :$   
 “[ ] projectors [of the type *c*]”  
 “[*c*] projectors [ ]”

Note that, the placement of the phrases before or after the main head ‘projectors’ says whether the phrase precedes or follows the head. (e.g. “projectors of the type *DLP*” or “*DLP* projectors”).

In the next entry we see simple joins being introduced in the elemental tuple expression.

$\{x|Projector(x) \wedge (\exists y)(Company(y) \wedge$

$x.manufacturer = y.id \wedge \psi(y)\} :$   
 “[ ] projectors [built by  $GEN(\{y|Company(y) \wedge \psi(y)\})$ ]”

The special term  $GEN(\dots)$  stands a description of a query where  $\psi(y)$  stands for an arbitrary tuple expression with the free variable  $y$ . Such a definition provides for relative clauses (e.g., “the projectors [built by [[Japanese] companies]]”).

Finally we note that very specific patterns of linguistic use may be authored into the phrasal lexicon. Consider for example:

$\{x|Projector(x) \wedge x.contrast > 5000 \wedge$   
 $x.brightness > 5000\} :$   
 “[high performance] projectors [ ]”

The above gives a detailed flavor of what entries look like. In practice however, administrators use a GUI-based authoring tool in which they initially name relations, attributes and joins, thus creating default entries. Since containment over our fragment tuple calculus is decidable, we can compile the entries into a subsumption hierarchy which organizes the phrasal lexicon for processing, inspection and extension. This hierarchy is instrumental in generating paraphrases, but only certain patterns are flagged as being used in generation, thus enabling asymmetric configuration, in which the language that can be recognized, is much broader than the language that may be generated. Finally the hierarchically organized phrasal lexicon provides a structure that administrators can navigate, edit and extend as they witness coverage leaks and non-fluent paraphrases in the history of user interactions.

## 4. ANALYSIS AND PARAPHRASING

Our approach to question analysis is to reformulate it as a classical state-space search problem. The initial state of this search is the input sentence and the goal state is a query expression that could have generated the input sentence. Intermediate states have a query that accounts for a prefix of the input sentence as well as the remainder of the sentence yet to be parsed. States also have an accrued *cost* which recaptures how much ‘fudging’ was required to match input words with phrasal patterns. Solutions exceeding a certain cost must be paraphrased back to the user for confirmation. Solutions exceeding an even greater cost are deemed complete failures. Semantically distinct solutions within a fixed cost of the best solution are presented as rival parses. Again, the presence of a theorem prover enables us to determine semantic equivalence.

The paraphrasing model of STEP [10] works by semantically sorting a tuple calculus expression into the subsumption hierarchy, fetching the patterns of its immediate parents, calculating unifying substitutions and combining patterns so that features agree. The resulting phrase consists of a set of modifiers, followed by a single head, followed by a set of complements. This process is complicated, however, by the fact that generation may be recursive.

## 5. DEMONSTRATION

Our demonstration aims to showcase the functionality of STEP across a variety of use cases. To accomplish this we will demonstrate STEP over a digital projector database, a student grades database and finally over a simulated database for the financial services help desk domain. Each of these interfaces shall be web accessible throughout the conference and afterwards. VLDB delegates will be welcome to try out the system to empirically determine its usefulness, scalability and robustness.

In addition to introducing the above web sites, we will also illustrate our web-based authoring tool which allows for the rapid configuration of STEP over arbitrary databases. It should be noted that a web-based STEP interface over a geography database [8] has been available since 2004. In this time we have had several thousand visitors pose geography queries and have used such feedback to guide system development.

## 6. SIGNIFICANCE

Since the late 80’s, the topic of NLI to databases has been somewhat dormant. While it is true that MicroSoft introduced the ENGLISHQUERY product in 1998, that product is now discontinued and in our opinion it ‘failed’ because it did not heed Codd’s advise to include a query paraphrasing mechanism. While some noteworthy efforts have recently addressed reducing NLI to database configuration costs [14, 12, 7], we believe that addressing issues of user acceptance (see section 1.1) outweigh the benefits of reducing configuration costs. In any case, STEP adopts an authoring tool approach to reducing configuration costs.

While the development of database-backed speech systems has been an active area, such systems typically map speech to very simple propositional slot-value lists; only recently have systematic efforts been undertaken to map to more expressive, though relationally incomplete, languages [3].

The introduction of a viable natural language interface to database technology would be a highly significant event for today’s web. Though our current focus is on natural language interfaces to relational databases, in future work may extend our approach to natural language access to semi-structured data via XQUERY/XPATH [7]. Still what really matters is to perfect our technology and designs so that visitors prefer our NLI because it provides significant advantages over forms and navigational interfaces.

## 7. ACKNOWLEDGEMENTS

Thanks to Peter Olofsson and Alexander Näslund for developing STEP’s web-based authoring tool and to Anna Hopfgarten for her authoring and design work.

## 8. REFERENCES

- [1] I. Androustopoulos and G. Ritchie. Database interfaces. In *Handbook of Natural Language Processing*, 209–240. Marcel Dekker Inc., 2000.
- [2] P. Bernays and M. Schönfinkel. Zum Entscheidungsproblem der mathematischen Logik. *Mathematische Annalen*, 99:342–372, 1928.
- [3] J. Boye and M. Wirén. Robust parsing and spoken negotiative dialogue with databases. *Natural Language Engineering*, to appear, 2007.
- [4] E.F. Codd. Seven steps to rendezvous with the casual user. In *IFIP Working Conference Data Base Management*, 179–200, 1974.
- [5] A. Copestake and K. Spärck Jones. Natural language interfaces to databases. *The Natural Language Review*, 5(4):225–249, 1990.
- [6] B. Grosz, D. Appelt, P. Martin, and F. Pereira. Team: An experiment in the design of transportable natural-language interfaces. *AI*, 32(2):173–243, 1987.
- [7] Y. Li et al. DaNaLIX: a domain-adaptive natural language interface for querying XML. In *SIGMOD*, 1165–1168, 2007.
- [8] W. May. Information extraction and integration with FLORID: The MONDIAL case study. Technical Report 131, Universität Freiburg, Institut für Informatik, 1999.
- [9] M. Minock. A phrasal approach to natural language access over relational databases. *NLDB*, 333–336, 2005.
- [10] M. Minock. Modular generation of relational query paraphrases. *Research on Language and Computation*, 4(1):1–29, 2006.
- [11] M. Minock. Natural language updates to databases through dialogue. *NLDB*, 203–208, 2006.
- [12] R. Mooney. Learning semantic parsers: An important but under-studied problem. In *AAAI 2004 Spring Symposium on Language Learning*, 39–44, 2004.
- [13] W. Ogden and P. Bernick. Using natural language interfaces. Technical Report MCCS-96-299, CRL, New Mexico State University, Las Cruces, 1996.
- [14] A. Popescu, O. Etzioni, and H. Kautz. Towards a theory of natural language interfaces to databases. *Intelligent User Interfaces*, 149–157, 2003.
- [15] D. Waltz. An English question answering system for a large relational database. *CACM*, 21:526–539, 1978.
- [16] C. Weidenbach et al. SPASS version 2.0. *Conference on Automated Deduction*, 275–279, 2002.