DESIGN FOR A FULLY TRANSPORTABLE NATURAL LANGUAGE FRONT-END TO DATABASE MANAGEMENT SYSTEMS

J. TERRY NUTTER, STEVE J. SAFIGAN AND ANGEL M. DIAZ, JR.

TR 89-20

DESIGN FOR A FULLY TRANSPORTABLE NATURAL LANGUAGE FRONT-END TO DATABASE MANAGEMENT SYSTEMS

J. Terry Nutter
Department of Computer Science
Virginia Tech
Blacksburg, VA 24061

Steve J. Safigan
Naval Surface Weapons Center

Angel M. Diaz, Jr.
Department of Computer Science
Virginia Tech
Blacksburg, VA 24061

Abstract

Natural language front-ends to database management systems represent a major improvement in accessibility for non-expert users. Unfortunately, such interfaces usually require extensive customizing not only of the front-end, but also of the data manager and hence of the DBMS itself. Developing such customized systems represents a huge investment of time and resources. In the 1980s, research has centered on making such systems portable at least across data bases, and in some cases across data base management systems. This report describes an architecture for complete transportability with minimal reprogramming, which has been partially implemented in a prototype system called TIPS. The TIPS architecture is compared briefly with other recent architectures, with attention to ease of portability, amount and locus of reprogramming needed, and extent of coverage both linguistically and in terms of database operations.

1. Introduction

Conventional database management systems (DBMSs) provide efficient storage and fast retrieval for large bodies of information. However, accesses and updates to such systems must normally be made in formal, highly structured query languages. Hence getting the information out of a database can prove a task not unlike programming, making it very hard for non-experts to use such systems. Motivation for adding

natural language interfaces to such systems are thus obvious.

Unfortunately, the realities of designing such systems often directly oppose those motivations. Broad coverage natural language understanding systems require huge bodies of information themselves. They are expensive both to develop and to run. By the time the interface has been paid for, many of the advantages of the DBMS have been obviated. Accepting a relatively narrower coverage of natural language helps somewhat; as a rule, natural language front-ends have been made economically feasible for individual databases by extensive customizing.

But while customizing makes a single front-end easier to develop and less costly to run, it makes the *marginal* cost of developing new front-ends very high indeed. Natural language interfaces will not be feasible for general use until that marginal cost decreases. In other words, for practical usability, front-ends must become readily transportable.

Two dimensions of transportability are important: transportability across data domains, and transportability across data managers. Transportability across data domains entails the ability to adapt the sublanguage of English which is accepted to deal with different kinds of information. Transportability across data managers entails translating queries into different target languages. Hence the first can be seen as flexibility in what can be accepted, and the second as flexibility in what can be done with the inputs once accepted. Systems which are transportable across domains but not across managers can only be used with a single DBMS, but can be used for many applications on that DBMS. Systems which are transportable in the second dimension can be used with a variety of DBMSs. We call front-ends which are transportable in both dimensions *fully transportable*. This report describes an architecture for a fully transportable natural language front-end to DBMSs.

2. Background

Interfaces which are transportable in the first dimension only, that is, across data domains, have been around for some years. In 1981, researchers at SRI reported on a prototype for transportability across domains, the Transportable English Datamanager (TED) [Hendrix and Lewis 1981]. Other such systems include PRE [Epstein 1985], Datalog [Hafner and Godden 1985], and IRUS [Bates and Bobrow 1983]. Domain transportable systems typically guarantee portability by one of two strategies. The first strategy, which IRUS, Datalog, and PRE use, isolates the portion of the semantics (and in Datalog's case of the syntax) which is domain-dependent, and reprograms at need. This requires changing the essential working code of the system, but usually in a limited and controlled fashion. The approach presupposes that porting across domains is the exception rather than the rule. The second strategy, used in TED, implements an interface expert which interviews the system administrator to produce the necessary domain-dependent information.

These systems do not address portability across data managers. On the contrary, in most cases, the interface is built into a customized data manager, with intimate access to the database and to database operations. In the case of TED, the coupling is so tight that it can be used to deal with many of the problems of term identification, although at the cost of restriction to a very tight subset of English.

ASK [Thompson and Thompson 1985] provides an interesting "missing link" between systems portable only in the first dimention and fully transportable interfaces. ASK itself is a data manager for a DBMS. However, it provides a facility for querying a foreign database. To do this, the user explicitly requests foreign access from ASK. ASK then calls an interview expert which queries the user on all information needed to access that system -- information ranging from the details of the foreign system's query language and database structure to physical details such as hardware types, baud rates, etc. When it has all this information, it translates the user's query into the foreign language, configures itself as a terminal, queries the foreign system, and dumps the result as unprocessed text.

This is ingenious, but it has an obvious disadvantage. To query a foreign system successfully, the user must know in detail about its hardware and software features and about how to formulate its queries. A user with that expertise could get the same information faster and more easily by walking to another terminal, logging onto the other system, and just asking for it.

ASK is so limited in its ability to deal with foreign data managers precisely because it has been designed with the opposite of second dimension portability in mind: it *is* its data manager. To achieve portability across DBMSs, interfaces must be decoupled as far as possible from the data manager. A recent system exhibiting this kind of portability is TEAM [Grosz, Appelt, Martin and Pereira 1987]. The architecture of the TEAM system is given in Figure 1.

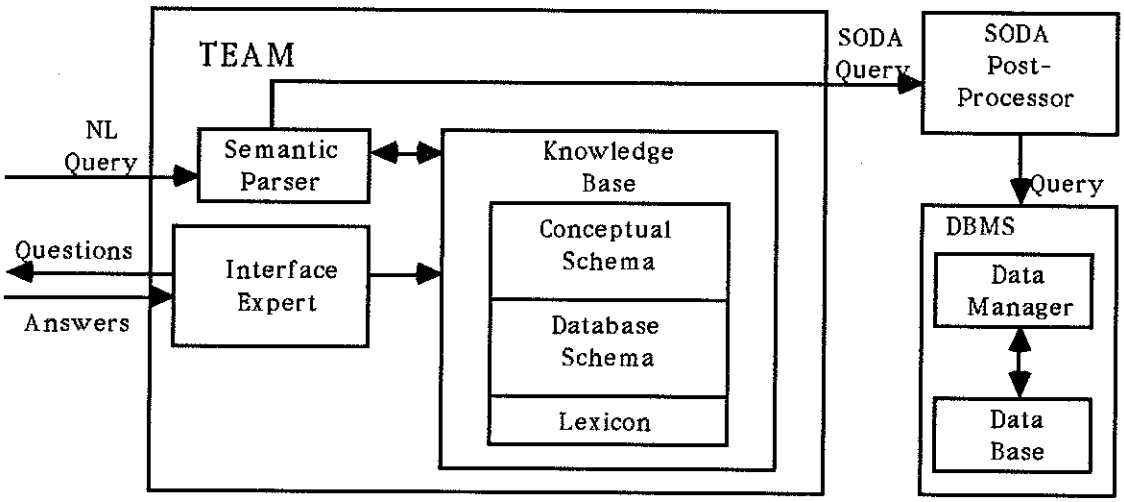

Figure 1. Architecture of TEAM

TEAM achieves transportability across domains by isolating a layered knowledge base containing information about the objects referred to in the data base and their relations (the conceptual schema), information mapping the objects and relations of the conceptual schema onto target database structures (the database schema), and information mapping the objects and relations into linguistic items used to refer to them (the lexicon). An interface expert interviews the system administrator to get the necessary information to customize this stratified knowledge base to a given data domain. So far, this is much the same as the structure of TED.

However, instead of translating the natural language query directly into a database query (in the language of the data manager to which it is adapted), TEAM takes a layered approach also to translation. Given a natural language query, TEAM translates the query into a parse tree, converts the parse tree into a logical form (extension of first order predicate calculus), then uses a schema translator to transform the logical form into an expression in an intermediate query language called SODA [Moore 1979]. A post-processor then translates the SODA query into a query in the language of the actual DBMS.

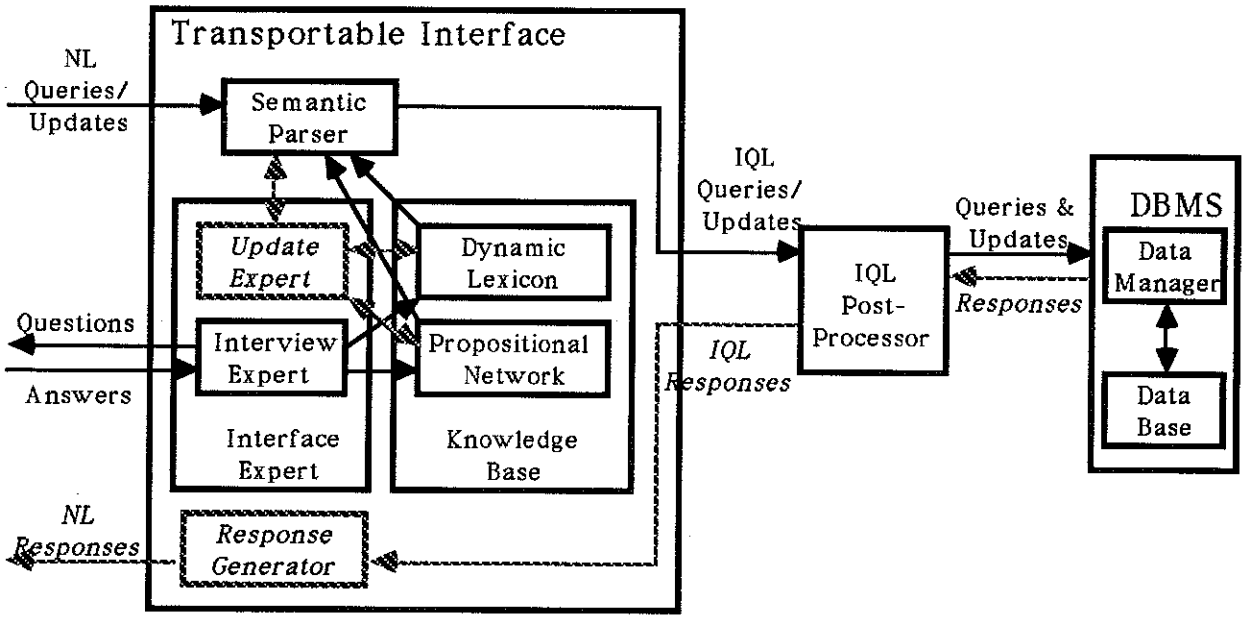

Figure 2. TIPS Architecture

3. Overall System Design

The approach taken here, and partially implemented in the Transportable Interface Prototype System (TIPS), is closely related to that embodied in TEAM. Our fundamental system design has the structure given in Figure 2. There are three major points of difference between this system and TEAM. First, the semantic parser takes a more direct route to a very simple intermediate query language. The less elaborate parsing procedure is designed to result in a faster parse. Second, the knowledge base in this design is more uniformly structured, allowing simpler access to its information and simple extension to more sophisticated methods (including some limited deduction). Third, the design presented here includes features for maintaining knowledge base and lexicon currency across alterations to the database by automating updates to the knowledge base as a direct consequence of additions to and updates of the database.

TIPS has three major subsystems: *a semantic parser*, *a knowledge base*, and an *interface expert*. Transportability in the first dimension, across data domains, is achieved through the knowledge base and interface expert, which jointly gather and store the domain information the parser needs. Transportability across DBMSs is

achieved by using an intermediate query language to represent parser output.

3.1 The Knowledge Base

To parse English requests into database queries, the system must be able to map English terms intelligently onto terms the database will recognize. This involves several different kinds of information which must all be available to the parser. First, it must have access to data definitions, including file names, field names, and field types. Without such information, the system cannot determine how the database query should be organized, given that many different organizations are possible. Second, it must have some way to determine corresponding synonyms and verbs for files and fields, and corresponding adjectives and units of measure for numeric fields. Third, key value fields must be identified for each file. Fourth, the links between database files must be specified to allow joins. Finally, to aid in disambiguation and establishing context for queries, it is very helpful to know the identity and contents of each field which modifies the data base file. (A modifying field is one whose contents describe, define, or classify the object of the database file.)

In TIPS, most of this information is obtained through the interface expert (see below), and stored in a dynamic lexicon and a propositional semantic network using SNePS [Shapiro 1979] [Shapiro and Rapaport 1986]. Further customization to the target data base can reduce problems of ambiguities; it may also be necessary to add some highly specialized contextual information, which only applies to a particular data base, to increase the scope of what is recognized by the parser. However, the information obtained by the interface expert alone is sufficient to allow a broad class of queries over most domains.

The use of SNePS has a number of advantages. First, it permits uniform representation of many different kinds of information, in a linked form for rapid access. Path-based access and reasoning in SNePS is very fast; all the knowledge about data objects is directly linked to information about the object hierarchies they belong to, the files they reside in, and so on. To date, we have found no case where the interface needs full first order inference (although that too is available); the path-based facilities have let the system exploit the associative nature of the representation to provide rapid semantic support for query parsing.

3.2 The Interface Expert

The TIPS interface expert consists of two components: the familiar *interview expert*, whose purpose is to query the system manager to determine the structure of an existing database, and the *update expert*, which the parser invokes in the course of parsing commands to add to or alter the database. In response to answers to queries, the interview expert adds terms to the lexicon and builds representations of the information the parser will need to parse queries and updates. The interview expert is based on a set of strategies for determining what terms can best be added relative to a given domain. These strategies are made available to the administrator, who is then interviewed to obtain the actual terms and structural information. The update expert performs the changes to the knowledge base to maintain consistency as alterations are made. So long as all updates and additions are performed through the interface, this insures consistency between the knowledge base in the interface and the actual database structure and contents.

3.3 The Intermediate Language and Semantic Parser

An intermediate query language to support transportability across data managers must have at least the following properties:

1. It must be unambiguous.

2. Its query specifications must correspond to and unambiguously determine the organization of the data in the target database.

3. It must not require operations on the data that the DBMS cannot perform or that cannot be *simply and systematically* performed as a result of multiple queries. Since the target DBMS is not know at the time of language design, it should require only operations that are routinely available on standard commercial DBMSs.

In short, the intermediate language query must capture all the information that will go into the formal database query, and do so in a way that makes it relatively trivial to get it back out. The cost of transporting the interface to a new DBMS is precisely the cost of developing a post-processor to transform intermediate language queries to target language queries. If that cost is high, then transportability has not really been achieved.

IQL is an extremely simple intermediate query language. It consists of non-empty lists of up to four registers: a *retrieval* register, a *qualification* register, an *operation* register, and a *join* register. The retrieval register contains a list of pairs of the form *<filename>.<fieldname>*, specifying what fields are to be retrieved from what files. The *qualification* register contains the clauses which serve to qualify or restrict the subject of the query; clauses allow field content comparisons (values of a given field less than or greater than a certain specified value), specification of maximum or minimum, and boolean combinations of qualifications using *not*, *and*, and *or*. The operation register determines the operation to be performed; valid operations include for instance *list*, *count*, *total*, and *yes/no*. The join register specifies links between files when more than one file is to be used. The information contained in these registers can be translated to formal queries in almost any standard commercial DBMS language using only simple syntactic methods.

The parsing portion of TIPS is based on a relatively simple ATN [Woods 1973] [Bates 1978]. The grammar is implemeted in the SNePS natural language processing facility, so that incorporating tests on the semantic network structure is trivial. The implemented prototype is not sophisticated; its coverage of English is relatively limited (though unlike TED, it does allow for a variety of verbs other than BE and HAVE, allowing and dealing with both transitive and intransitive structures), and it has little semantic component beyond the ability to use the information in the knowledge base.

4. Status of the Prototype

The current implementation of TIPS does not include the update expert or

related systems; a very preliminary version of these portions has been implemented, but not yet integrated. In addition, we have not implemented the response generator; all responses at present come raw from the DBMS. (In general, anything on Figure 2 that is labeled in italics or marked with dashed lines is not implemented.) It does, however, include a fully implemented interview expert, a simple semantic grammar, and a knowlege base. The system was tested at least two databases on each of two different target DBMS languages. The data domains included a personnel system and a sports information system (on boxing). The targets languages were Model 204 DBMS, a fairly standard relational database language, and the McMax system, a highly interactive database management tool developed for use in the Macintosh environment.

The Model 204 post-processor was developed first. Our experience with transporting it to McMax was particularly heartening because of the major differences in approach between the two and the specific problems which the design of McMax presented. McMax was designed for ease of interactive use. It was never anticipated that a software system would be sending it programs. Several of the features which contribute to its ease of use interactively make it particularly nasty to deal with in a post-processor situation. A primary example is the lengths necessary to deal with joins.

McMax is window oriented. Each file is opened in its own workspace. Only one workspace is active at a time, and while one is active, information cannot be retrieved from any other. The only means for passing information across workspace boundaries -- other of course than looking at it on the screen -- is to store it in a database variable. This means that to perform a join, the system must activate all the necessary workspaces, select a variable, enter the first workspace, find what to put in the variable, enter the second, figure out what to do with the variable, and so on. Easy if you are sitting in front of the screen, but a nuisance to program in a post-processor. Our post-processor does not currently handle arbitrary joins, but it can handle any request in its language which involves only two files, and the extension to more than two files is tedious rather than hard.

We of course encountered similar problems with other language features. Model 204 has numbered lines, and any command may refer to any other by line number. This facilitated post-processing greatly. McMax has no similar feature. (People using McMax aren't really expected to program it. They are expected to do one relatively simple thing at a time, and get a lot of information in front of them to keep them from having to work out complex queries.) Hence the McMax programs corresponding to relatively simple ordinary queries turn out to be logically far more complex than the corresponding Model 204 ones. This accounts for most of the difference in post-processor length.

The body of the prototype and the Model 204 post-processor were implemented by a Masters level graduate student. The port to McMax was performed by a senior level undergraduate student carrying a full time course load. When he began the project, the undergraduate had never seen an ATN, never worked with natural language or with databases, and never done substantial programming in LISP. Less than three months later, the post-processor was complete and running. A full-time professional familiar with the techniques and architecture could probably have completed the port in under a week.

The prototype system covers a reasonable range of normal queries; more impressively, it does so with minimal overhead for transportability. The entire post-processor to transform IQL into Model 204 takes about six letter-size pages of LISP

code; the post-processor for McMax is about twice that long. The interview to establish the knowledge base takes about twenty minutes (and in the full system version, would only need to be done once in the history of any given database). The prototype implementation is far from a production level system, but we feel it suffices to show the viability and advantages of this approach. Sample runs are given in the appendix.

5. Proposed Extensions

The most obvious set of proposed extensions to the system involves completing the implementation of the update expert and of the natural language generation module (the currently unimplemented parts of the system architecture diagram). At present, we are returning answers as they come from the DBMS. There are obvious and simple techniques for improving on this, ranging from simply echoing a declarative version of the original question, with the answer filling in the (final position) gap, to more sophisticated techniques for smoother responses. These are points we simply haven't gotten around to yet. In addition, there are several ways in which we would like to enhance existing features.

5.1 Interactive Parsing

At times in parsing it may be useful to ask for the user's help to resolve ambiguity. When a query to the knowledge base reveals an ambiguous mapping which the system can't resolve, the system could ask the user to resolve it. For instance, if the user asks "How old is John F. Kennedy?", and the system knows of three John F. Kennedys (the man, the airport, and the ship), it can ask the user which is meant. In database queries, such unresolvable ambiguities should be rare enough that this is an enhancement and not a nuisance. An interactive parse approach is in fact taken in EUFID [Templeton 1979].

5.2 Enhanced parser coverage.

At present, our system manages conjunctions poorly at best. The general problems of interpreting conjunctions are well known, but the parser can do better than it does simply by looking for what syntactic category of parallel constructs it can find on both sides. Similar remarks apply to disjunction. In general, our current parser is syntactically primitive; we look forward to extending it.

5.3 Enhanced lexicon management

The current design is largely limited to vocabulary expressly provided through the interface expert. One obvious improvement would involve including information about English at the knowledge base level, so that, for instance, if the system knew that "fast" was a corresponding adjective for the database attribute "speed", it could work out that so is "quick". A relational lexicon of the kind reported in [Nutter 1989], [France, Fox, Nutter and Chen 1989] and [Nutter, Fox and Evens 1989] has the potential greatly to extend the interface's linguistic flexibility with little to no processing overhead. The full lexicon described in those reports is extremely large; it remains to

be seen how useful a smaller lexicon (say restricted to the 2000 most common words) would prove in this context.

6. Conclusions

The keys to achieving full transportability lie in (1) having a local knowledge base which adequately and accurately reflects the organization of the database, and (2) producing output in an intermediate query language with a low post-processor cost. Enhancing the knowledge base and tuning the intermediate query language are therefore important ways to improve transportability. We have attempted several things in this direction.

First, we have designed an extremely simple intermediate query language, whose translation cost (into at least one actual DBMS language) is negligible both in terms of development and in run time. Second, we have designed a knowledge base which is uniform and expressively powerful but which allows rapid access to most of the important and frequently needed information. Third, our design allows not only for intial customizing to a database but also for automatically maintaining consistency over the database lifetime. The solution presented provides a kernel system for fully transportable natural language access to standard commercial database management systems.

References

Bates, M. "The Theory and Practice of Augmented Transition Network Grammars." In *Natural Language Communication with Computers*, L. Bolc., ed., Springer-Verlag (Berlin) 1978.

Bates, M. and R. Bobrow. "A Transportable Natural Language Interface," *Proc. Sixth Ann. Conf. of the ACM SIGIR*, ACM (New York) 1983.

Epstein, S.S. "Transportable Natural Language Processing through Simplicity — The PRE System," *ACM Trans. Office Inf. Sys.* 3 1985.

France, R.K., Fox, E.A., Nutter, J.T. and Chen, Q.F. "Building a relational lexicon for text understanding and retrieval," *Proceedings of the First International Language Acquisition Workshop*, Detroit, Michigan, August 21, 1989.

Grosz, B.J., D.E. Appelt, P.A. Martin and F. Pereira. "TEAM: An Experiment in the Design of Transportable Natural-Language Interfaces," *Artif. Intel.* 32 1987.

Hafner, C. and K. Godden. "Portability of Syntax and Semantics in Datalog," *ACM Trans. Office Inf. Sys.* 3 1985.

Hendrix, G.G. and W.H. Lewis. "Transportable Natural Language Interfaces to Databases," *Proc. 19th Ann. Meeting of the ACL*, 1981.

Nutter, J.T. "Representing knowledge about words," *Proceedings of the Second National Conference of AI and Cognitive Science*, Dublin, September 14-15, 1989.

Nutter, J.T., Fox, E.A. and Evens, M.W. "Building a lexicon from machine-readable dictionaries for improved information retrieval," *The Dynamic Text: 16th ALLC and 9th ICCH International Conferences*, Toronto, Ontario, June 6-9, 1989.

Shapiro, S.C. "The SNePS Semantic Network Processing System." In *Associative Networks: Representation and Use of Knowledge by Computers*, N.V. Findler, ed., Academic Press (New York) 1979.

Shapiro, S.C. and W. Rapaport. "SNePS Considered as a Fully Intensional Propositional Semantic Network," *Proc. AAAI-86*, 1986.

Thompson, B.H. and F.B. Thompson. "ASK is Transportable in Half a Dozen Ways," *ACM Trans. Office Inf. Sys.* 3 1985.

Woods, W.A. "An Experimental Parsing System for Transition Network Grammars." In *Natural Language Processing*, R. Rustin, ed., Algorithmics Press (New York) 1973.