

**THE ECONOMICS OF NATURAL LANGUAGE INTERFACES:
NATURAL LANGUAGE PROCESSING TECHNOLOGY
AS A SCARCE RESOURCE***

Sumali J. Conlon**

John R. Conlon***

and

Tabitha L. James**

**Division of MIS

E-mail: sconlon@bus.olemiss.edu

Phone: 662-915-5470

***Division of Economics

E-mail: jconlon@bus.olemiss.edu

Phone: 662-915-5708

School of Business Administration
University of Mississippi
University, MS 3867

June, 2002

* The authors wish to thank Martha Evens, Jon Greenblatt, Steve Hyde, Taha A. Kass-Hout, Penda Tomlinson, Richard Wood, and William Woods for very helpful discussions. An anonymous referee also made extremely valuable suggestions, in particular, encouraging us to interview commercial NLI developers and customers. Sumali Conlon would also like to acknowledge support from the University of Mississippi Faculty Small Grant Program, the Hearin Foundation, and the Office of Naval Research. Any remaining errors are, of course, ours.

**THE ECONOMICS OF NATURAL LANGUAGE INTERFACES:
NATURAL LANGUAGE PROCESSING TECHNOLOGY
AS A SCARCE RESOURCE**

Abstract: This paper discusses appropriate application areas for a Natural Language Interface (NLI) to a database. This requires comparing an NLI with competing approaches to interfaces, including (a) other types of user-friendly interfaces, such as icon, menu, or Query By Example (QBE) systems, and (b) training of users so that they become comfortable with less user-friendly interfaces. Also, since NLI technology is still limited, users may need to learn how to use NLI's themselves. Network externalities may therefore arise, where users who learn how to work with one NLI become more comfortable using other NLI's. This suggests that NLI popularity may snowball at some point.

On the technology side, appropriate application domains must be sufficiently well structured to allow for feasible implementation. However, if the application domain is extremely well structured, other user-friendly interfaces such as menus, become more cost effective. We consider a simple prototype NLI (one currently under construction at the University of Mississippi), to illustrate when an NLI can achieve levels of flexibility that are unattainable by menu, report, or QBE-based systems. We conclude that, since current NLI technology is still limited, considerable application-specific customization is necessary. Approaches to customization are discussed, as are some currently existing commercial NLI's.

KEYWORDS AND PHRASES: Natural language interfaces, database management systems, network externalities, economics of information systems.

1. Introduction

Natural Language Interface (NLI) technology has become increasingly sophisticated in recent years (Woods 1978, Waltz 1978, Wilensky et al. 1984, Grosz et al. 1987, Adam et al. 1994, Allen, 1995, Androutsopoulos et al. 1995, Ein-Dor and Spiegler 1995, Nanduri and Rugaber 1996, ELF, 2001b). However, practical application of these technologies has remained limited. This paper asks why applications have not been more widespread, and also what approaches can potentially make NLI technologies more applicable in the future.

One way to understand the problem is to ask whether NLI applications have been uncommon (a) because it is difficult to build a useful NLI, or (b) because it is easy to get along *without* an NLI (Russell and Norvig, 1995, p. 694). A determination of the appropriate application areas for NLI's therefore requires a careful analysis of substitutes for NLI's.

There are two major alternatives to an NLI. The most obvious is some other type of user-friendly interface, such as a menu or report-based system. A second, less obvious alternative is the training of potential users to handle less user-friendly interfaces, such as spreadsheets, statistical packages, or programming languages such as C++. This suggests that NLI's are more appropriate if it is difficult to train the users to work with less user-friendly systems (Adam et al. 1994, Ein-Dor and Spiegler 1995). Also, since NLI technology is still limited, training in how to use limited NLI's *themselves* may substitute for better NLI technology. NLI popularity may therefore snowball at some point, as users become more comfortable with NLI's, so more organizations implement NLI's, and so on. Indeed, it is possible that *current* technology would become very successful if users were sufficiently familiar with existing NLI's.

On the other hand, an NLI is more expensive to develop than a menu or form-based

interface. This cost has two components: the up-front cost of creating a sophisticated natural language understanding system, and the organization or application-specific costs of customizing the NLI for the application areas needed by actual users.

A basic natural language understanding system represents essentially a sunk cost. Once a basic system is developed, it can be reproduced at very low cost. Unfortunately, it is so difficult to build a sophisticated system, that all existing systems still face serious limitations (though see Section 8 on some systems which are already commercially available). These limitations may be partly circumvented by customizing systems for more narrow application domains. This, however, involves significant application-specific costs.

Moreover, even a carefully customized system will be ineffective unless the problem domain is sufficiently well structured. However, if the domain is *very* well structured, other user-friendly interfaces, such as menu bases, may be more cost effective. Thus, the domain should be well structured, but not too well structured. We examine a typical prototype NLI (one currently under construction at the University of Mississippi), to examine whether an NLI can achieve levels of flexibility that are unattainable by menu, form, or report-based systems.

The next section considers an NLI as a substitute for user training in more difficult interfaces. It also considers the users' need to learn how to use NLI's themselves. Section 3 examines implications of NLI cost structure. Section 4 shows how the recursive structure of natural languages such as English facilitate NLI technology. Section 5 describes a prototype NLI currently under construction at the University of Mississippi, and Section 6 uses this system to illustrate the relative challenges faced by an NLI versus simpler interface technologies. Section 7 considers customization of NLI's to a specific organization's needs, and also considers

ways to economize on organization-specific customization costs. Section 8 discusses existing commercial NLI's, focusing on two: ELF and EasyAsk, and Section 9 concludes.

2. The Substitution between an NLI and User Training

An NLI allows an organization to substitute a sophisticated interface for the training of the users of the system. Thus, an NLI is more appropriate if it is difficult to train the users to work with less user-friendly systems, such as spreadsheets, statistical packages, or programming languages like C++ or SQL. This is discussed in Section 2.1. On the other hand, existing NLI's are highly imperfect. Thus, user training in how to work around the limitations of NLI's themselves may also substitute for better NLI technology, as discussed in Section 2.2.

2.1 Factors affecting user training costs

Five considerations affect users' training costs: (1) the number of potential users, (2) the opportunity cost of the time of these users, (3) the complexity of the tasks these users perform with the system, (4) the prior skills of these users, and (5) the users' relation to the organization.

First, a sophisticated interface such as an NLI is more appropriate, the larger the number of potential users. If a system has few users, then it may be cheaper to train these users to work with a less user-friendly interface. On the other hand, with many users to train, an NLI can reduce these training costs (Adam et al. 1994, Ein-Dor and Spiegler 1995). The same effect occurs if there is a high rate of turnover among employees using the system.

Second, the opportunity cost of the users' time affects training costs. Training takes time and effort. If users are upper level executives, for example, then it may be a costly waste of their

time to train them in less user-friendly interfaces. This would make NLI's more appropriate. In fact, this factor may often override the first consideration above. That is, it may make more sense to adopt an NLI for a few upper level executives as compared to many lower level employees, because of the higher cost of time for the upper level executives.

Third, with simple tasks, users should need very little training to work with menu-based interfaces, say. For example, hotel clerks need to work with a limited set of forms to make reservations, check in guests, etc. This requires very little training. By contrast, a mid or upper-level manager may need flexible access to the database in order to prepare ad hoc reports, and so, may require more extensive training. Of course, the sophistication of an application should also increase the cost of building an NLI (see Section 6).

Fourth, if users are highly trained, then their training will frequently include skill in using the computer software supporting their specialization. For example, users doing sophisticated statistical analysis will usually know how to use statistical packages such as SAS or SPSS. These employees will therefore not need an NLI to communicate with the system. However, employees with less technical training, such as some upper level managers, may need an NLI to access the statistical models developed by the employees with more technical training.

Fifth, if the user is not a formal member of an organization, then it may be difficult for the organization to train the user to work with an interface. This is because learning how to use a non-user-friendly interface is a relation-specific investment on the part of the user. If the user is not a member of the organization, then her relation with the organization is less stable, so she has less incentive to incur such relation-specific costs (see Klein et al. 1978, or Williamson, 1981).

This becomes especially important in web-based applications. For example, an NLI

might be valuable in business to consumer (B2C) web pages. Since customers have brief and sporadic relations with the firm, they will not learn how to use an interface unless it is very user-friendly. This problem is aggravated by the large number of customers that access a typical web page, and the lack of prior technical training of the typical customer.

A second example may be in web-based applications aimed at *business to business* (B2B) communication across firms in the supply chain. For example, suppose that a firm wants to allow executives from firms upstream or downstream in the supply chain to access information from the firm's database through the web, say (subject to appropriate security constraints). Then, since supply chain partners are not formal members of the organization, they may be less willing to learn how to use a non-user-friendly interface. An NLI may therefore be more appropriate. An NLI might also be useful for two of the other reasons discussed above: the number of such users may be large, with nontrivial turnover, and the opportunity cost of these users' time may be high if these users include higher level executives of the supply chain partners' firms.

In summary, an NLI will be more appropriate the larger the number of potential users, the higher the opportunity cost of the time of these users, the less computer-related specialized training these users already have, and the more tenuous users' relations to the firm. The complexity of the task will also increase user training costs, but there may be an offsetting increase in the cost of building an NLI sophisticated enough to handle the task.

2.2 Learning how to use an NLI, and implications for network externalities

The discussion so far has ignored the costs of learning how to use a *natural language*

interface. However, these costs may be nontrivial, and also may have interesting implications. Existing NLI's can handle only a limited range of natural language input, and it is generally not obvious to the user what queries the system will and will not understand. This confusion can take two forms (Androutsopoulos et al. 1995). First, users might be overly optimistic about an NLI's capabilities, and become frustrated as NLI's fail to respond as expected to complicated queries. Alternatively, users may be overly pessimistic, so they fail to explore the full range of the NLI's capabilities. Thus, users generally need some practice with an NLI before they become familiar with the range of natural language queries the system can handle.

For users who are members of an organization, formal training may therefore be useful. For example, system administrators might create a set of sample questions, covering the users' most frequent query types. Then, as users become more confident with the system, they might be encouraged to explore the full capabilities of the system, with oversight from the technical support staff (see Section 8.1 below for an example involving the commercial NLI ELF).

For users who are not members of the organization, training may be more difficult. One challenge may be presented by B2C Internet applications. Internet retailers cannot expect customers to devote much effort to learning how to use their NLI's. Users may therefore fail to take full advantage of the NLI. For example, according to Richard Wood of the e-commerce NLI firm EasyAsk, customers tend to type very short queries into the EasyAsk NLI, even though EasyAsk can handle longer, more precise queries. Thus, customers must undergo a self-training period before they learn how to make full use of a B2C NLI.

This would seem to create a major problem for B2C NLI use. However, if the NLI's of different retailers have similar linguistic capabilities, then customer experience with one

retailer's NLI may have positive spillovers to other retailers' NLI's. That is, B2C NLI's may generate positive externalities (Shy, 1995, Chapter 10, Shy, 2001, Shapiro and Varian, 1999).

There are at least five possible ways to cope with these externalities. First, an Internet retailer may take advantage of these externalities by adopting an NLI similar to the NLI's of other retailers. Customers who have learned how to use other retailers' NLI's will then be able to use this new retailer's NLI more effectively. For example, a retailer may adapt the NLI system which is most popular among other Internet retailers.

Second, brand naming by NLI developers themselves may facilitate this process. For example, both Talbots and Wine.com feature the EasyAsk logo on their product search web pages. Thus, users familiar with other web pages incorporating the EasyAsk NLI will be better able to use the NLI's on Talbots and Wine.com's web pages. However, other EasyAsk customers, such as Lands' End and Coldwater Creek do not currently display the EasyAsk logo.

Third, pricing by NLI developers may help to internalize this externality. If an NLI developer benefits from consumers learning how to work with its system, then it also benefits from retailers who adopt its system earlier. It should therefore give discounts to these early adopters, and so, bear some of the costs of this early adoption. Of course, it should require display of its logo in return, so consumers know whose NLI they are learning about.

Fourth, if the vendor can convince Internet retailers that consumers are becoming familiar with its NLI, more Internet retailers will adopt it, so more consumers *will* learn how to use it.

Finally, a vendor might make direct efforts to help consumers become familiar with its NLI interface. For example, it may give Internet retailers discounts if they include drop-down

windows of sample requests that users can ask of the NLI. NLI vendors might also use focus groups to learn the types of requests users would ask if they understood the full capabilities of the NLI, and include these requests in the drop-down windows.

The network externalities of NLI's in B2C e-commerce also suggest that these NLI applications may be subject to sudden, unexpected growth spurts. Initially, customers may become frustrated with these systems. However, as they become familiar with the strengths and limitations of these systems, these systems may become increasingly popular among users, and so, increasingly profitable to Internet retailers. In fact, it is theoretically possible that, even with current, *existing* technology, NLI applications may become dramatically more popular as users become familiar with them. As more users learn how to work with NLI's, more firms may adopt them, leading still more users to become familiar with them, creating a snowball effect.

B2B applications may call for a different approach. First, to the extent that relationships between supply chain partners are more stable than relationships between consumers and Internet retailers, supply chain partners may be willing to learn how to use one another's NLI's. Firms in an industry may also choose to follow a formal or informal industry standard. If all firms in the auto industry, say, worked with the same NLI provider, then that provider could standardize the NLI's capabilities so that a parts supplier familiar with GM's supply chain NLI would have a pretty good idea how to work with Chrysler's supply chain NLI. That is, in B2B applications, firms may be better able to internalize network externalities.

In summary, even given the limited nature of current NLI technology, existing NLI's may become more popular as more users undertake the initial cost of learning how to work around these limitations. In fact, it may be a sign of the growing success of NLI systems when

their limitations become a popular topic of conversation.

3. Implications of Cost Structure--Dynamics, Competition, and Pricing

Like all software, NLI's have a cost structure involving a large initial fixed cost (the cost of developing the initial software), and a small variable cost (Shapiro and Varian 1999). This cost structure has three major implications: (i) a long period of development will be needed before truly reliable NLI's are built, but once the quality of the software crosses this threshold, applications may grow rapidly, (ii) firms must receive an average price per unit significantly above marginal cost in order to justify their initial investment, so a firm will only invest in the development of NLI software if it expects to have considerable market power once the system is developed, and (iii) this market power, combined with low marginal cost, means that NLI vendors will face nontrivial pricing problems. These three implications, however, are modified by the particular characteristics of NLI's, as explained below.

Natural languages are much more complicated than formal languages (Ein-dor and Spiegler 1995). The up-front cost of developing workable NLI's is therefore very large, so it will be a while before truly versatile NLI's are developed. Thus, there may be a long period involving clever adaptations of limited NLI systems to specific areas. These clever adaptations may require significant organization-specific customization to transport these systems to new application domains (Martin et al. 1983, Epstein 1985, Grosz et al, 1987). See Section 7 below.

This imaginative use of limited NLI technology may, in turn, create a market for improved natural language processing (NLP) systems, and so, may speed up the development of such systems. Once this technology crosses a threshold level, adoption should then accelerate.

This increased demand should then stimulate more rapid development in a snowball fashion, especially since these systems can be replicated at very low cost. This snowball effect reinforces the effect of the network externality, mentioned in Section 2.2. Thus, potential users should expect rapid improvements, once these technologies begin to mature.

Next, low cost distribution conflicts with the initial developer's incentive to incur the fixed cost of developing the system. A developer will only develop a system if she expects to sell it for significantly more than the marginal cost of producing an additional unit, since she will otherwise be unable to cover the up-front fixed development cost. Thus, imitation or the threat of parallel development of NLP systems by competing firms may reduce each firm's incentives to devote resources to developing a sophisticated system. This yields a classic argument for the patent system, and for public support for the development of technology (Shy 1995, Chapter 9, Tirole 1988, Chapter 10). Put another way, firms will only incur the fixed costs of developing sophisticated systems if they do not expect markets to be perfectly competitive. Otherwise prices would be driven down to marginal cost, and so, would be much less than average cost. The firm could not then make a profit. Anticipating this, few firms would enter the market in the first place, so perfect competition would never be achieved (Shapiro and Varian 1999).

On the other hand, since NLI applications usually involve considerable customization, this may reduce post-entry competition. First, customization leads to product differentiation. NLI vendors might differentiate themselves from each other by their ability to customize for different industries, and an NLI vendor's team of programmers, who do the customization, may further differentiate the vendor from its competitors. Second, competition may also be affected by the capacity constraints imposed by the need to customize systems. If systems require

significant customization, then a firm's ability to expand is limited by the personnel it has available to customize systems for individual buyers. This, in turn, can reduce the aggressiveness with which firms compete (Kreps and Scheinkman, 1983).

In any case, competition will generally not be perfect, so firms retain significant control over prices. Pricing then becomes a nontrivial issue. In particular, since marginal cost is much less than average cost, it is optimal for vendors to sell "marginal" units for much less than the average unit. NLI vendors will therefore want to pursue some sort of price discrimination. For example, vendors will want to sell NLI systems to small firms for much less than they charge large firms. If small firms are "marginal" buyers, then vendors want to charge them a price closer to marginal cost, and so, much less than the prices it charges to larger firms. For example, software vendors generally give implicit discounts to small firms by offering per-user licenses.

On the other hand, in terms of pricing for a single customer, the NLI vendor may want to charge some sort of analogue of a quantity discount. That is, rather than charge a simple per-user fee, the vendor might charge an "installation fee" for the basic setup of the system, but then charge a smaller fee for marginal improvements to the system. The installation fee would presumably be larger for larger organizations, but, for any given organization, the fee for marginal improvements would be close to marginal cost. Given the customer's decision to adopt the system, this pricing structure would reduce the distortion in the customer's decision of how *extensively* to use the NLI within the organization. This would maximize the overall gains from trade, which the vendor could then extract through the fixed fees (Jagpal, 1999, Chapter 5). The vendor would only want to charge a marginal per unit price significantly above marginal cost if the vendor felt that a customer's demand for extra features revealed a higher overall willingness

to pay (Oi, 1971, Maskin and Riley, 1984, Tirole, 1988, Chapter 3).

The customer should also be able to negotiate marginal improvements in the system at close to marginal cost, unless such requests revealed too much about its willingness to pay.

Since NLI's will be more useful to some users than for others (see Section 2.1), this should also affect the fixed fee. For example, an NLI may be very valuable to an executive with a high opportunity cost of time and a wide range of ad hoc questions to ask. By contrast, NLI systems may offer some convenience value for hotel clerks, but will presumably offer much less value per user, since a form or menu based system will be almost as good. Thus, it would make sense for NLI vendors to base fees on the *type* of user, as well as the size of the firm. In the related industry of voice recognition, for example, the company Kurzweil charges much more for systems aimed at doctors than for systems aimed at general users (Shapiro and Varian, 1999, p. 60). Prices could also depend on the types of customization requested by the users.

Customization of NLI's also implies lock-in once a system has been customized. Thus, customers will tend to renew licenses with the same NLI vendor. This reduces competition for established customers, so vendors may be tempted to charge established customers a lot for license renewals. This may make customers reluctant to commit to a system unless the vendor can commit *itself* to not exploit this lock-in too aggressively. If the vendor cannot commit itself in this way, it may need to offer a low initial price, to compensate for this lock-in.

Finally, as mentioned in Section 2.2, NLI's may experience significant network externalities, where users who become familiar with one website's NLI, say, may be able to work better with other websites' NLI's, so NLI vendors may have an incentive to provide price discounts to early adopters of their NLI systems. Firms are offering similar discounts when they

give away beta versions of their product. Vendors can potentially use similar discounts to internalize some of the network externalities among users of their systems.

4. How is NLP Technology Feasible?

This section now turns to the technological challenges faced by NLI's versus simpler interfaces. Since natural languages are so complicated, it may seem impossible to develop viable NLP technologies. The set of possible sentences may seem too vast to handle.

What makes NLP potentially feasible is that natural languages use their rules recursively. That is, it uses a limited set of rules over and over to generate an unlimited set of sentences. We can illustrate this by considering a different natural language, which computers can handle very well, i.e., mathematics. Suppose, e.g., that one wants to determine the present value of a project with three years of income, Y_1 , Y_2 , and Y_3 , and initial outlay C . Then one must calculate

$$\frac{Y_1}{1+r} + \frac{Y_2}{(1+r)^2} + \frac{Y_3}{(1+r)^3} - C.$$

Performing a calculation like this on a menu-based system would be very difficult. However, people use programming languages and spreadsheets to work with formulas like this all the time. The key is that the user and the computer share the language of mathematics. Computers can work with mathematics because mathematics is a recursive language which uses simple components to build up more complicated structures. For example, the formula above is a combination of the operations of addition, subtraction, multiplication, and division.

Of course, ordinary natural languages are much more complicated than mathematics (Allen 1995). Nevertheless, natural languages have a recursive structure, known as phrase

structure (Chomsky 1957, Baker 1995). Linguists represent this structure using parse trees. For example, the question “Who teaches the students in the class” has a (simplified) parse tree

```
s(noun(“who”),verb_pp(“teaches”,noun_p(“the”,”students”), pp(“in”,noun_p(“the”,“class”))))
```

This indicates that the main verb of the sentence is “teaches.” Also, the subject of the verb is “who,” and the object of the verb is “the students.” Finally, the prepositional phrase (pp) indicates that the teaching is occurring “in the class.” This structure is known as a parse *tree* because it can also be represented as a tree structure.

The recursive nature of English allows the system to use the same grammatical rules over and over. Thus, the noun phrase rule is applied here to both the noun phrase “the students” and the noun phrase “the class.” This reusability is analogous to the reusability of arithmetic operations, exploited in programming languages and spreadsheets. Just as a program can combine a limited set of operations to generate an unlimited set of formulas, a parser uses a limited set of grammatical rules to interpret an unlimited set of English sentences.

Thus, NLI’s derive a benefit over menu or form-based interfaces, because NLI’s are able to exploit the recursive structure of natural languages such as English to process a wider range of user needs. Moreover, an NLI can draw on a much wider vocabulary than a menu or form-based interface can. This vocabulary will generally be stored in a lexicon (Conlon et al., 1993), for the system’s use, and is an additional advantage of an NLI over a simpler interface type.

5. A Prototype NLI System

The system at the University of Mississippi uses the programming language PROLOG to analyze English sentences and translate these sentences into SQL. The application domain we focus on is a very simple version of a university database. The following describes the different components of this system: the parser, the lexicon, the knowledge base, and the SQL generator.

1) The Parser: As discussed above, the parser breaks complicated sentences into small parts with their grammatical roles, e.g., parts of speech, indicated (Woods 1970, Covington 1994). An example is given in the previous section. Our parser also moves "WH-words" (such as "who," "which," "when," "where," etc.) to the appropriate position in the sentence to allow the SQL generator to determine what the user really wants to ask. For example, the sentence

"What did bob teach in H110"

is parsed as:

```
s(noun("bob"),verb_pp("teach",noun("what"),pp("in",noun("H110"))))
```

This indicates that "what" is asking for the object of the verb "taught." Again, the fact that such movement rules can be combined with phrase structure rules in an unlimited number of ways enhances the flexibility of an NLI. Our parser builds on the approach in Covington (1994).

The parser, then, is the heart of the typical NLP system, and should be fairly independent of application area. That is, the parser should not be organization-specific. For more on parsers,

see Covington et al. (1988), Covington (1994), and Allen (1995). For references on the syntactic theory underlying parsers, including movement, see Chomsky (1957) and Baker (1995).

2) The Lexicon: Our lexicon contains the system's vocabulary, and indicates the part of speech for each word. This vocabulary includes both the words in the database, and the words that the users might employ. Some sample entries in the lexicon are:

wh("who").	wh("which").	wh("where").
do("do").	do("did").	do("does").
d("the").	d("a").	d("an").
n("mis409").	n("class").	n("computer").
v("is").	v("teach").	v("take").
aj("red").	aj("higher").	aj("small").
p("in").	p("on").	p("under").

Thus, "who" is a wh-word, "does" is a conjugation of the important auxiliary verb "do," "the" is a determiner, "computer" is a noun, "teach" is a verb, "red" is an adjective, and "in" is a preposition. The parser uses this classification to determine the recursive structure of sentences.

The size of the lexicon can be fruitfully contrasted with the size of a typical menu base. The effective language of a menu base is limited by the number of options offered at a typical level, raised to the number of levels in the menu base. For example, a menu base with 8 options per level and three levels will offer roughly $8^3 = 512$ options, giving a vocabulary of about 512 choices. Any reasonably large lexicon will therefore dwarf most menu bases. Again, this underscores the increased flexibility of an NLI as compared to a menu base. Note, however, that if each menu item yields a flexible form-based interface, such as a query by example interface, this may increase the menu base's flexibility. Section 6 compares NLI's to more sophisticated user interfaces, such as column-selection form or report wizards.

Finally, lexicons should be fairly “transportable” from one organization to another, particularly within application area (Martin et al. 1983, Epstein 1985, Grosz et al. 1987). This suggests that markets could develop for lexicons in different application areas. This would allow an organization to purchase the underlying NLI system and the lexicon separately, though the organization may also want to be able to modify the lexicon in various ways. Area-specific lexicons are useful because, even if a word has several meanings in general, only a few of these meanings may apply to a given area. This may significantly reduce ambiguity.

3) The Knowledge Base: The knowledge base allows the computer to relate the words in a user’s query to the terms in the database. The following are some sample entries in our system’s knowledge base.

isa(“professor”, “rank”).	isa(“mis”, “c_title”).
isa(“mis”, “major”).	isa(“art”, “department”).
isa(“classes”, “table_name”).	isa(“computer”, “tool”).
v_obj(“take”, “c_title”).	v_obj(“taught”, “c_title”).
actor(“takes”, “stuname”).	actor(“teaches”, “facname”).
in_table(“course_no”, “classes”).	in_table(“dept”, “faculty”).
in_table(“rank”, “faculty”).	in_table(“credits”, “student”).

This part of the system is needed to interpret the words in a user’s query in terms of the attribute names, etc., in the database. Thus, if the user’s query includes the word “professor,” the system will know that “professor” is a “rank,” and will know to look for the “rank” attribute in the “faculty” table. The attribute names in the database provide the ultimate meanings of the key words in a user’s query. The knowledge base provides a bridge between the words employed by the user and these ultimate meanings given by the data base.

Since these ultimate meanings are given by the application (in this case, the database),

the organization employing the NLI must custom-build much of the knowledge base for the specific application. This may often be a major part of the cost to the organization of adopting an NLI. Breaking off this information into a separate knowledge-base component should make this customization easier. However, certain sophisticated commercial NLI's can already do much of this customization automatically. See the discussion of ELF's Analyzer in Section 8.1 below.

Also, much of the knowledge in a knowledge base will not be organization specific. For example, the entry isa("faculty", "employee") would apply across universities. Such entries are called "lexical semantic relations" (Ahlsweide and Evens 1988, Wang et al. 1985), and a great deal of work has been done building large sets of these relations. For example, WordNet (Miller 1990, 1995, Fellbaum 1998) is a large collection of lexical semantic relations built at Princeton University. WordNet builds on six major lexical semantic relations: synonyms, hypernyms (X is a kind of Y), hyponyms (Y is a kind of X), holonyms (X is a part of Y), meronyms (Y is a part of X), and familiarity. WordNet contains lexical semantic relations between 95,600 different word forms. Ultimately, markets may develop for area-specific knowledge bases.

4) The SQL Generator: The SQL generator takes sentence structures in the form of parse trees, and combines them with information from the lexicon and knowledge base to generate commands that database systems can understand (SQL in the case of our system). The generated SQL statements are then executed by database management systems (DBMS) to produce answers for the users. For example, the question:

"Who teaches mis in H110 at 10:00"

is parsed as

```
s(noun("who"),verb_pp("teaches",n("mis"),
                        pp("in",noun("H110")),pp("at",noun("10:00")))).
```

The SQL generator then converts this parse tree into an SQL statement as:

```
SELECT    facname,
FROM      faculty, classes,
WHERE     c_title = 'mis'
and       room = 'H110'
and       sched = '10:00'
and       faculty.facid = classes.facid
```

Since the SQL generator is a separate component, the system can be more easily transported between different DBMS's. In a transportable version of the system, the database administrator would need to indicate which DBMS the interface will be used with, so the SQL generator generates the appropriate dialect of SQL (Martin et al. 1983, Epstein 1985, Grosz et al. 1987).

6. When is a Problem Domain "Well Structured but not too Well Structured?"

We now use the general structure of our NLI to consider how well structured an application domain should be in order for an NLI to be the appropriate form of interface. We

focus on applications which require a reasonably flexible interface, for example, a mid-level executive's task of extracting data from a database to prepare an ad hoc report.

To assess the flexibility of an NLI, we compare an NLI to a "column-selection" system, such as Oracle Developer's Data Block Wizard (Morrison and Morrison 2002). Such systems allow users to custom-build forms by selecting from lists of columns, and so, gives users considerable flexibility, without requiring them to know SQL. Thus, we ask whether it makes more sense for an organization to customize an NLI for their particular application domain, or expect users to work through a column-selection process to build a form or report. In the process we consider the demands placed on the language understanding system and on the organization-specific knowledge base which would have to be developed for a typical application. Throughout we use the University of Mississippi NLI as the representative system, though we occasionally refer to more sophisticated commercial systems.

We proceed in five stages. First, we consider a query which requests information from columns of a single table. Second, we consider constraints on individual columns in a single table, for example, listing employees with salaries greater than \$40,000. Third, we allow for cross-column constraints (e.g., within-major GPA greater than overall GPA). Fourth, we consider queries requiring joins. Finally, we consider the problem of performing various computations on columns, given various own and cross-column constraints. For example, what is the average salary of employees with at least five years experience with the firm?

For each of these tasks, we consider the relative difficulty of handling the task with an NLI versus a sophisticated non-NLI interface, such as Oracle/Developer's Data Block Wizard. How convenient is an NLI versus such a column-selection system in handling each task, and

how does this convenience vary with, e.g., the number of columns in the database? This will allow us to determine which tasks, if any, may be sufficiently simple that they can be handled by an NLI, but too complicated to handle by even a sophisticated menu-based system.

6.1 Selecting columns of a single table

First consider a query such as

“What are the salaries of all faculty?”

This query is parsed as

```
s(noun(“what”),verb_phrase(“are”,d_n1_pp(“the”,”salaries”,
                                     pp(“of”,av_n(“all”,“faculty”)))))).
```

The SQL generator then converts this parse tree into the SQL statement

```
SELECT    facname, salaries
FROM      faculty
```

What does the NLI need to do to make this conversion? First, it needs to recognize that “what” refers to “salaries.” This follows because “what” is the subject and “salaries” is the object of the verb “is.” This does not involve any organization-specific knowledge, so the ability

to make inferences like this can be incorporated into the base system. This therefore represents a fixed cost component of developing an NLI, and so, in the long run, should be relatively inexpensive to the organization (Tirole 1988). Next, the *organization-specific* knowledge the system needs in the knowledge base for this query includes:

(1) “Salaries” is a column name in the faculty table. This is represented by the rule `in_table(“salary”,“faculty”)` in the knowledge base.

(2) The column `facname` in the faculty table is specified as a special naming column for the faculty table. Thus, whenever the system presents any information from the faculty table, it also presents out the corresponding element in the `facname` column. This is done for the convenience of the user in interpreting the answer.

Note that a different organization might list all employees’ salaries in a large “employee” table, for example. For this second organization, “salaries of the faculty” might be obtained by selecting the “salaries” column from the “employee” table, where “employee_category” equals “faculty.” Thus, since different organizations generally use different database structures, they must indicate where they locate data in the database, requiring considerable preparation by the system administrator. Significant organization-specific customization must therefore go into a successful NLI application. On the other hand, some NLI systems do a lot of this customization automatically. See the discussion of the ELF NLI in Section 8.1 below.

By contrast, consider what a user has to do to perform this query through a column-selection system, such as Oracle/Developer’s Data Block Wizard and Layout Wizard (Morrison and Morrison 2002) or Microsoft Access’s Form Wizard and Report Wizard (Adamski et al. 2000). Using the Oracle/Developer system, for example, the user first selects a table, say, the

Faculty table, and selects the facname and salary columns, using the Data Block Wizard. She then uses the Layout Wizard to display the information using either a Form or tabular format.

This column-selection process is not completely trivial, but it is not prohibitively difficult for a user with a moderate amount of training. On the other hand, if it is difficult to provide even a small amount of training, as in B2C and B2B applications (see Section 2), then the column-selection process may be infeasible, increasing the relative benefit of an NLI.

The convenience of an NLI versus a column-selection system also depends on the size of the database. As the number of columns in the database grows, an NLI might save the user the effort of scrolling down long lists of columns. The typical database for a medium sized company contains about two hundred tables, and about thirty columns per table, yielding as many as 6000 columns that the user must look through. This may become prohibitively time consuming. On the other hand, a database with a large number of columns may yield frequent ambiguities for the NLI. For example, the word “price” could refer to the price of an input purchased by the organization, or the price of a product or service sold by the firm.

Finally, it may be possible to combine natural language processing capabilities with a column-selection or menu-based system. For example, the system could ask the user to provide keywords to indicate which issues she is interested in, and the system could return a more limited list of columns related to those keywords (this approach was suggested by a referee). That is, the system could function something like an information retrieval system. The user would then select columns from this shorter, more focused list to create a form or report, as above. The system could even incorporate a thesaurus or knowledge base, to expand the list of columns retrieved. This may provide an interface which is easier to build than an NLI, but easier

to use than an unaided column-selection system.

6.2 Restrictions on individual columns

Next, consider the query

“What does Smith teach in H110 at 10:00?”

This query is parsed as

```
S(noun(“smith”), verb_pp2(“teach”, noun(“what”),  
                           pp(“in”, noun(“H110”), pp(“at”, noun(“10:00”))))
```

The SQL generator then ultimately converts this into the SQL statement:

```
SELECT    facname, course_no, c_title  
FROM      faculty, classes  
WHERE     facname = ‘Smith’  
and       room = ‘H110’  
and       sched = ‘10:00’  
and       faculty.facid = classes.facid
```

Ignore the last line for the moment.

What does the system need to know in order to handle this query? In terms of the knowledge base, the system uses the following rules:

(1) The wh-word “what” appears in the object position of the verb “teach.” This triggers the rule `v_obj(“teach”,“c_title”)`, and so, indicates that the question may be asking for an entry from the column `c_title`, which contains course titles. Note, incidentally, that this builds on the movement facility of our parser, which moved the wh-word “what” from the subject position in the original question to the object position in the parse tree.

(2) The rule `isa(“smith”,“facname”)` indicates that one restriction on the `facname` column will be `facname = ‘smith’`.

(3) The rule `isa(“H110”,“room”)` indicates that a restriction on the `room` column will be `room = ‘H110’`.

(4) The rule `isa(“10:00”,“sched”)` indicates that a restriction on the `sched` column will be `sched = ‘10:00’`.

Thus, building a knowledge base capable of handling this query would again require a considerable amount of organization-specific work, though much of it could be done with a sophisticated install procedure.

How would this compare to processing by a column-selection system such as Oracle/Developer’s Data Block and Layout Wizards? First, the user would select the five columns `facname`, `course_no`, `c_title`, `room`, and `sched`, as above (ignore for the moment the fact that the columns actually come from two different tables). If the user chooses the tabular format, she would then use a query by example (QBE) approach to specify query constraints by typing ‘smith’ in the `facname` column, ‘H110’ in the `room` column, and ‘10:00’ in the `sched` column.

This procedure would require somewhat more knowledge of the Oracle/Developer Wizard facility. However, with a moderate amount of training, an employee of the firm, say, could easily use such a system unless the number of columns was prohibitively large. Thus, in this case, a column-selection approach might be as flexible as an NLI, and easier to build.

On the other hand, the number of columns accessed will tend to be larger if the user wants to put restrictions on columns. For example, in the above query, only the facname, course_no, and c_title columns are wanted, but the room and sched columns are used to impose restrictions (ignoring, for the moment, the joining condition in the last line of the SQL statement). Thus, in this case, the user would need to locate five columns, even though only three are ultimately desired. Therefore, if the number of columns in the database is large, then a column-selection system will again tend to become somewhat more difficult to use.

6.3 Restrictions across columns

Next, consider the following query, which involves a comparison between two columns:

“Which students have overall GPA less than major GPA?”

This query is parsed as

```
S(noun(“Which”,“students”),verb_pp(“have”,compp( np_adj(adj(“overall”),n(“GPA”)),
comp(“less”,“than”),np_adj(adj(“major”),n(“GPA”))))).
```

The SQL generator then ultimately converts this into the SQL statement:

```
SELECT    stuname
FROM      student
WHERE     overall_gpa < major_gpa
```

The new element, here, is the comparison of two columns, `overall_gpa` and `major_gpa`. This requires more of the parser, i.e., a comparison construction (`compp` and `comp`). Also, organization-specific elements in the knowledge base are necessary to allow the system to find “overall GPA” and “major GPA” in the database, though this is needed even setting aside cross-column restrictions. Beyond that, the SQL generator needs to recognize “less than” as “<” in SQL, but again, this can be prespecified in the SQL generator. Thus, cross-column restrictions seem to create few extra difficulties for an NLI.

By contrast, imposing this cross-column restriction in a column-selection system such as Oracle/Developer’s Data Block and Layout Wizard is nontrivial. In the Oracle system, the user must first create the basic form structure, as above. She must then select the “property palette,” and type the cross column restriction `Major_GPA > Overall_GPA` into the “where clause” row. This requires considerable sophistication of the user. Such problems may therefore be so poorly structured that an NLI would be preferred to a column-selection approach. The comparative advantage of NLIs may therefore increase at this point. On the other hand, it is not clear how often users want to use such cross-column restrictions in practice.

6.4 Joining Tables

The joining of two tables is obtained by forming the Cartesian product of two tables and imposing a cross-column restriction. The major new element is that the system must know which $\text{Column_X} = \text{Column_Y}$ restrictions are needed to perform the join. This will be trivial for both types of system if each table has at most one foreign key pointing to any other table. Sometimes this may not be the case. For example, a film industry database may have foreign keys in the “movie” table for both “lead actor” and “supporting actor,” pointing to the same “actor” table. In this case, the problem of joining would become nontrivial. This difficulty, however, would be common to both systems. Note that the query in Section 6.2 above involved joining two tables, and so, required the cross-column restriction $\text{faculty.facid} = \text{classes.facid}$.

If we move beyond flat files to full-fledged normalized databases, the number of columns can begin to explode, as mentioned above. Using a column-selection interface could therefore become very time consuming, especially for people who are not familiar with the database. The user would have to look at many tables, and many columns per table. The system will then only be able to provide the correct answer if it is clear what joining conditions to use to join the tables.

6.5 Performing computations on columns

Consider the query

“What is the average GPA of MIS majors?”

After parsing, the system generates the SQL commands

```
SELECT      Avg(overall_gpa)
FROM        student
WHERE       major = 'mis'
```

Here the knowledge base needs to know that there is no column such as `average_gpa` in which “average GPA” could be found, but that, instead, “average GPA” would request that the `Avg` function be applied to the `overall_gpa` column, after applying the `major = 'mis'` restriction. Likewise, in a column-selection system, the user can restrict the entry in the `major` column to be `mis`, and then specify that an average of the `overall_gpa` column is desired. There does not seem to be any special advantage to an NLI in this case.

6.6 Other types of queries

More sophisticated NLI's can understand question structures more complicated than those handled by the University of Mississippi system. For example, the commercial NLI, ELF (ELF, 2001b, [SQL Server 7.0 link](#)) can handle questions such as “Which customers have ordered both Konbu and Filo Mix?” This question, referring to Microsoft's Northwind database, requires the system to select customers based on having “Konbu” in one row of that customer's product column, and “Filo Mix” in another row of that same customer's product column. Similarly, the query “Show company names of the suppliers that have more than 3 products,” requires ELF to select suppliers whose supplier ID's appear in more than three rows of the

product table of the Northwind database. These queries would be very difficult to handle using a column-selection system, but can be handled by sophisticated NLI's such as ELF. Thus, NLI's like ELF have a significant comparative advantage over column-selection-based systems for questions like this.

7. Customization Issues

As mentioned above, current natural language processing technology is still very limited, so NLI's in the near future will depend heavily on careful application-specific and organization-specific customization, especially for the systems' knowledge bases and lexicons. This customization represents the major component of the variable cost of extending NLI's to new users. A central question then becomes how to economize on these customization costs.

Part of the solution is the development of sophisticated installation systems, like ELF's Analyzer, which can perform part of this customization automatically (see Section 8.1). A second way to economize on customization costs is to transform these costs, as much as possible, from variable organization-specific costs into fixed costs, which can be spread over many organizations. This can be done by customizing in two stages: first, build partly customized industry-specific or application-specific systems. This would involve refining the lexicon, knowledge base, and SQL generator for specific application areas. For example, in the related area of voice recognition, the company Kurzweil sells different versions of its product, with vocabularies appropriate for different areas, such as law, medicine, etc. (Shapiro and Varian, 1999, pp. 59-60).

However, to the extent that the knowledge base and SQL generator interprets words in

terms of an organization's specific database columns and entries, detailed knowledge of the organization's database needs to be built into the knowledge base. The above *partly* customized systems would therefore have to be *further* customized for specific organizations within an industry. While sophisticated install systems like the ELF Analyzer can help with this, NLI's will nevertheless remain somewhat expensive to adapt by individual organizations.

This, in turn, suggests *standardizing organizations' database structures* as well as their knowledge bases and lexicons. This is actually being done by the software companies mentioned in Section 8.1 below. For example, the company MD Computer Consultants was already marketing a software product for doctors' offices, which included a carefully designed database structure. Since they have already designed this standardized database structure for their application area, it is much easier for them to customize the commercial NLI, ELF as an interface to their databases. They can then sell the interface, jointly with the database structure, to many customers. This allows their customers to trade off some flexibility in terms of database design, in exchange for a cheaper and more accurate NLI. A similar strategy is used by the human resource software company, also discussed in Section 8.1. The existence of these firms suggests that an "NLI infrastructure" is developing, of firms which build customized NLI-database systems for specific application areas.

The major role for organization-specific customization also suggests that the growth of NLI systems will require a labor market to develop, of people with skill in customizing knowledge bases and lexicons. Just as MIS programs currently turn out graduates who can handle sophisticated DBMS packages such as Oracle, Informix, and DB2, in the same way, future MIS programs may turn out graduates who know how to identify needed elements to

include in an NLI's customized knowledge base or lexicon.

This suggests that a network externality may exist in the spread of NLI technology (Shy 1995, Chapter 10, Shy, 2001, Shapiro and Varian 1999). NLI technology may depend upon professionals with skills in customizing NLI systems, but professionals will not develop these skills until NLI technologies become popular. This network externality has implications similar to those of the network externalities mentioned in Section 2.2, and the fixed-cost element in NLP systems, mentioned in Section 3. Adoption will initially be slow but, as NLI use achieves a critical mass, the area may begin to grow more rapidly, as professionals develop the skills to customize NLI's, and organizations adopt systems which draw upon these skills.

NLI vendors can internalize some of these externalities by providing customer support, to help organizations customize their NLI. Both ELF and EasyAsk do this (Section 8). Oracle used a similar strategy in the early stages of the relational database market. Then, as the number of MIS personnel familiar with relational databases increased, Oracle was able to reduce the amount of direct customer support needed for users to successfully adapt their software (Read, 1999).

8. Commercially Available NLI's

A handful of major commercial NLI's have been developed in the past twenty years or so. The following briefly describes some NLI's, which are either currently available, or have been available in the past. Two commercially available systems, English Language Frontend (ELF), and EasyAsk, are then described in greater detail. Given the state of flux of the NLI field, the following discussion does not attempt to be exhaustive.

One of the first NLI products was Q&A, developed by Symantec. Symantec sold Q&A from 1982 to 1998, but has since discontinued Q&A, and turned to products related to Internet security, etc. Recently Lantica Software LLC has announced plans to develop a Q&A-compatible database interface, though it is not clear what its NLI capabilities will be.

A second major commercial NLI, Intellect, was developed for mainframes in the early 1980's, by Larry Harris, a former professor of Computer Science at Dartmouth. Harris went on to develop English Wizard, for PC's, in the mid 1990's, and EasyAsk for search of e-commerce sites and other enterprise databases (see EasyAsk, 2001, and below).

A third NLI, LanguageAccess, was briefly distributed by IBM in the early 1990's (Androutsopoulos et al., 1995, Language Industry Monitor, 1993), and a fourth, English Query, has been introduced by Microsoft, for use with Microsoft's SQL Server (Microsoft, 2000).

Finally, the NLI ELF (English Language Frontend) was introduced by the ELF Software Company in 1995. Previously, the same company introduced ELLIE, an NLI for dBase, in 1989 (see Elf, 2001a, and below).

To get some sense of the commercial implications of NLI's, we interviewed people at two commercial NLI companies, the ELF Software Company, and EasyAsk. We also interviewed three ELF customers, to get a user's perspective on the product. Unfortunately, it is beyond the scope of this paper to do a more systematic survey of NLI users.

8.1 The ELF NLI

The founder and CEO of ELF Software is computational linguist Jon Greenblatt. Greenblatt developed the basic interface in the mid 1980's, using transformational grammar

modeled after Chomsky's (1965) *Aspects* model. This part of the system involves about a thousand linguistic rules. Unlike the simple prototype discussed in Section 5 of the current paper, Greenblatt's system is not written in Prolog, but in Delphi, since Delphi allows greater control over the detailed workings of the computer. At this point, ELF has been marketed primarily for smaller databases, with a single-user edition costing only \$139 and a deluxe developer's edition costing \$499 (ELF, 2001c). Technically, the system is designed to work with larger and more complicated databases, as well.

The ELF system is much more sophisticated than the system in Section 5. It thus gives some sense of the advantages of sophisticated NLI's over their closest non-NLI competitors such as column-selection systems. For example, ELF Software presents a demonstration application on their website (ELF, 2001b, [SQL Server 7.0 link](#)), interfacing with Microsoft's Northwind sample database. They illustrate ELF with queries such as "Which customers have ordered both Konbu and Filo Mix," mentioned above. As argued above, a column-selection system would have difficulty answering a query like this. ELF, on the other hand, generates an extremely complicated SQL statement in response to this query, which gives back the correct answer. The fact that ELF can handle this query suggests how sophisticated some commercial NLI's are.

The ELF demo provides drop-down menus illustrating about forty queries. It also allows visitors to type in other queries over the web. For example it was able to handle typed-in questions such as "Which customers ordered tofu but not konbu?" and "Which customers ordered products with unit price greater than \$40?" The ELF demo is not perfect, however. For example, in response to the query "What is the price of tofu?" it gives shipping freight, not price. However, the response table clearly labels the shipping freight column as "Freight," so there is

little danger of user confusion. Also, in response to “What is the *unit* price of tofu?” (emphasis added), the demo returns the correct answer. Finally, ELF is designed so that a system administrator can easily fix problems like this, though Greenblatt says that he did very little customization with the Northwind demo. In summary, the ELF system, while not perfect, is able to handle a remarkable range of linguistic input and SQL output.

An aspect of ELF that Greenblatt takes special pride in is the install feature, called the Analyzer. ELF is designed to analyze the structure of a user’s database automatically, to determine which columns and tables different concepts will refer to. For example, in the Employees table of the Northwind data base, the ELF Analyzer identifies column names such as “Home Phone” as concepts that a user might ask about. Thus, ELF is able to do a considerable amount of customization automatically. However, the system administrator will generally also have to do some additional customization, such as defining synonyms for the different database concepts, to broaden ELF’s vocabulary.

To get a user’s perspective on ELF, we interviewed three ELF customers, Taha Kass-Hout, Penda Tomlinson, and Steve Hyde. Kass-Hout adopted ELF for an application at a public health institution, while Tomlinson and Hyde are incorporating ELF as an NLI into software products which their companies are selling.

Taha Kass-Hout adopted ELF to help fourteen people he was supervising, who had responsibility for inputting data and answering ad hoc questions concerning a relational database about medical information. These users had highschool degrees and a little beyond. A few had some experience with computers, and “knew how they thought,” according to Kass-Hout.

Installing and running the ELF Analyzer took Kass-Hout about an hour. Building the

additional vocabulary then took a few hours. Kass-Hout also developed a set of forty or fifty sample questions, whose structure the users could follow. This took Kass-Hout five or six days on and off, with users helping by telling him the things they wanted to ask. Most of the users were not sufficiently confident to experiment with different kinds of questions, and so, stayed close to Kass-Hout's original list of sample question structures.

A few of the users, however, began exploring the full range of what they could ask through ELF, using trial and error. They would try a question, and ELF would either indicate that it did not understand the question, or respond with something the user did not understand. For the first few weeks of this experimentation period, they would ask Kass-Hout about almost every new question. Eventually, however, they became confident with the system, and only called occasionally for reassurance. It helped, however, that, since the users had entered the data themselves, they were familiar with the structure of the database, so they could recognize when ELF gave an incorrect answer. In the end, Kass-Hout felt that the users who had gone through this exploratory process were "extremely happy with the results of ELF as an NLI."

Perhaps the most interesting thing about Kass-Hout's experience is the effectiveness of trial and error as a training tool. This suggests that a similar trial and error process might be an important step in users becoming comfortable with other NLI's, as argued in Section 2.2 above.

The other two customers we interviewed were incorporating ELF into systems for resale. Penda Tomlinson works for a company which supplies software systems to support the human resource function, and is incorporating ELF into these systems as a database interface tool. Their system currently has a complex report writer, and Tomlinson is hoping that ELF will allow users to create reports without knowing any SQL. Their target users are human resource

professionals: administrators and staff, with a focus on users at the executive level. The number of users per organization ranges from one to thirty. The basic database structure Tomlinson's firm supplies involves about eight tables. One table has about 100 columns, and the others have about 30 columns each. Customers can also ask for additional columns.

The fact that the firm designs the database gives these databases a lot of common, standardized, structure. This makes it easier to adapt the NLI to this structure, as argued in Section 7 above. However, ambiguity remains a problem, according to Tomlinson. The database's vocabulary is large, and different columns often have similar names, such as "contracted hours per day" versus "contracted hours per week." A major challenge in customizing ELF for their database product is to handle these ambiguities. One approach they are using is to give the user extra information in the case of ambiguity. For example, if the user asks a question about something "contractual" the system may return everything with "contractual" in it, and let the user sort it out.

Tomlinson initially tried his system out in-house (on colleagues in his firm). His sense is that, at the present time, when a user first starts using the system, without any sample questions for guidance, their queries have a success rate of about 50%. If given sample questions to illustrate the system, and one to one and a half hours of practice, their success rate rises to 70-80%. Some users are very passive, and accept whatever answer the system gives. Others, however, are more active, changing the wording of a question to see how the system responds. "We've had users randomly cut words out to see what happens" according to Tomlinson. "Once people feel like they can joke about it and feel comfortable with the system, then it's successful."

Tomlinson's firm has just begun trying the system on focus groups. Queries asked by

members of the focus groups include “Show telephone number of male employees aged over 18 with salaries over 25000,” “how many UK european employees are located at head office?” and “how many employees are in each department of each sex?” Note that these queries are very complicated, and presumably depend upon the full natural language understanding capabilities of a sophisticated NLI like ELF, and also on the customization done by Tomlinson’s firm. On the other hand, it is not clear how the queries will change as the system moves from focus groups to actual applications.

The third ELF customer we interviewed was Steve Hyde, at MD Computer Consultants, a supplier of software solutions for medical office applications. They are incorporating ELF as a database interface into Medical Office Mate, a scheduling and billing system for doctors’ offices. The database component of Medical Office Mate has about 36 tables, with about 520 columns that they want ELF to handle. This database contains information about patients, appointments, insurance companies, services delivered, payments received, and so on.

Hyde believes that the ELF NLI is very “robust.” He also feels that the Analyzer is very convenient, and does a lot of the customization work. Nevertheless, his company is doing a lot of additional customization to incorporate ELF into Medical Office Mate. Hyde feels that, since his company has designed the underlying standardized database structure, it is a lot easier for them to customize ELF to work well with that structure. This is again consistent with the argument in Section 7 above. He also maintains that careful normalization of the database reduces the problem of ambiguity. Rather than having a lot of columns with similar information in them, a well normalized database has less redundancy, and instead has foreign keys pointing to the location of the relevant data elsewhere in the database. He also feels that splitting the

database into views will reduce the ambiguities that ELF will have to deal with, though this will be less necessary if the database is designed well.

Hyde has a very high standard for what he hopes ELF will do, once incorporated into his product. He disagreed with the suggestion that users should be willing to experiment with the system in order to learn how to work around its limitations. He felt that his users will “just want the data so they can go home. They’re not going to want to play around with it. ... A successful NLI should save you work. It’s not a toy.” Nevertheless, he felt that beta testing is an important part of the development process. They plan to give the beta version, free, to “certain doctors with very knowledgeable staff.” These first few customers would presumably have a lot of complaints about the system. Then, as the system is refined in response to their complaints, he expects to have “a pretty good system.” Once the final version is ready for sale, the company will also give the new version to the beta users for free. Hyde felt that a reasonably accurate NLI would be significantly easier to use than a column-selection wizard/QBE system.

8.2 The EasyAsk NLI

EasyAsk serves two major types of applications: NLI’s for on-line retail web catalogues, and enterprise applications for searching heterogeneous internal databases. EasyAsk’s e-commerce customers include Lands’ End, Coldwater Creak, Wine.com, and Talbots.

From the user’s point of view, these NLI’s operate a lot like information retrieval (IR) systems. Users type in a request, and the system returns a list of catalogue items, much like a standard search engine returns a list of web pages. Internally, however, the system operates as a standard NLI, parsing the request and connecting its elements with the different database

concepts. This makes the system more intelligent than a standard IR system. For example, the system can handle a request such as “cotton sweaters between \$50 and \$60.” This question involves the prepositional phrase “between \$50 and \$60,” and requires the system to determine that the numbers refer to the price column in the database, even though the word “price” does not appear in the request. Typical questions for EasyAsk’s commercial sites range from “golfwear” to “petite red cotton polo shirt for less than \$30.” Note that this second request requires recursive application of the adjectives “cotton,” “red,” and “petite” to the compound noun “polo shirt,” before appending the prepositional phrase “for less than \$30.”

According to Richard Wood, vice president of development, the original installation of EasyAsk usually takes “a couple of weeks to a month.” The system then goes on line. For the first few months after the system goes live, EasyAsk is actively involved in reviewing the site and refining the site-specific lexicon. Ultimately, however, EasyAsk expects a system administrator on the customer’s side to take responsibility for the lexicon. The system keeps a log of users’ requests, with how the system translated the requests, and whether the system returned any answers. There is also an EasyAsk NLI to the log file, which allows the system administrator to ask questions such as “which requests most often failed to return answers.” The monetary cost of the three-year site license is “low six figures and up,” according to Wood, with daily consulting fees for the installation and a maintenance fee of 18%.

EasyAsk also builds enterprise applications, for search of internal databases. These applications generally involve multiple databases, and unstructured data such as documents. They are therefore more complicated, because the system must initially determine which databases should be queried, and then must integrate the returned information into a coherent

response. These systems must therefore combine standard IR techniques with NLI techniques. At this point, EasyAsk's commercial sales are primarily in e-commerce, with some enterprise applications, but they hope to expand the enterprise side of their business in the future.

9. Conclusion

We are now in a position to compare NLI's with simpler interfaces. First, we can see that an NLI depends on sophisticated processing components such as the parser. This is a disadvantage, because it is extremely difficult to build systems that can handle a very wide range of sentences. However, this represents a fixed cost which should, in the long run, have little effect on the prices of such systems. However, with existing systems, there is always a risk that a parser will fail to parse a sentence, or come up with the wrong interpretation.

In addition, since NLI technology is still limited, organizations must carefully customize the lexicon and knowledge base for their application domain. However, sophisticated install features can help with this customization, and "NLI infrastructure" firms are beginning to market standardized NLI-database systems to different types of users. Finally, since NLI's are still highly imperfect, users may have to learn through trial and error how to work around the limitations of existing NLI systems.

In this paper we have primarily considered NLI's and other types of user-friendly interfaces as competitors. However, it is also possible to combine these two types of interfaces. For example, as discussed in Section 6.1, a natural-language-based information retrieval capability might be combined with a column-selection procedure to help the user search through long lists of columns in creating a form or report. Other ways to combine NLI's with menu

bases also suggest themselves. For example, if a database is very big, one could break the database up into different views, and build different NLI's customized for those different views. These different views, in turn, could then be accessed through a menu. This may reduce the level of ambiguity each of these NLI's must deal with, and so, make these NLI's more accurate and easier to develop. Alternatively, if an NLI has trouble disambiguating a query, it may present a menu of answers appropriate to different interpretations of the query, and let the user choose between them. This is something like what the Ask Jeeves search engine does in a different context.

While the current paper has focused only on database applications, the relative advantage of NLI's may grow as we moved to more sophisticated systems, such as model bases or expert systems. Of course, with such sophisticated applications, the adaptation of the NLI may be even more complicated. It may therefore turn out that applications, such as information retrieval, requiring *less* robust natural language processing, may be more appropriate (Salton 1989, Jacobs 1992). An example of this might be Ask Jeeves. Finally, a third set of applications would involve voice recognition systems and certain types of graphic user interface (GUI) systems (we thank Martha Evens and William Woods for this suggestion). With voice recognition systems, the most natural language for a user to use would be, unsurprisingly, a natural language. While point and click technologies favor menus, voice recognition systems favor ordinary human language. Thus, as voice recognition systems progress, the value of systems which are able to interpret ordinary natural language sentences will increase. NLP technology is even used in voice recognition technology itself, since a lexicon is needed to translate a string of sounds into words, and parsing helps to disambiguate between various possible strings of words all of which

approximate the sounds picked up by the system. Finally, GUI systems are being developed which draw on voice recognition, NLP, and point and click technologies. These systems allow users to give commands like “I would like to add that [click] and that [click],” to facilitate extremely natural and flexible modes of communication to control computers.

References

- Adam, Nabil R., Arryya Gangopadhyay, and James Clifford, 1994. "A Form-Based Approach to Natural Language Query Processing." *Journal of Management Information Systems* 11(2), 109-135.
- Adamski, J. J., K. T. Finnegan, and C. Hommell, 2000. *Microsoft Access 2000*. Course Technology, Cambridge, MA.
- Ahlsweide, T. E., and M. Evens, 1988. "Generating a Relational Lexicon from a Machine Readable Dictionary." *International Journal of Lexicography* 1(3), 214-237.
- Allen, James, 1995, *Natural Language Understanding*, Second Edition. Benjamin/Cummings, Menlo Park, CA.
- Androutsopoulos, I., G. D. Ritchie, and P. Thanisch, 1995. "Natural Language Interfaces to Databases—An Introduction." *Journal of Natural Language Engineering* 1(1), 29-81.
- Baker, C. L., 1995, *English Syntax*, Second Edition. MIT Press, Cambridge, MA.
- Chomsky, N., 1957, *Syntactic Structures*. Mouton: The Hague.
- Chomsky, N., 1965, *Aspects of the Theory of Syntax*. MIT Press, Cambridge, MA.
- Conlon, S., Evens, M., Ahlsweide, T., & Strutz, R., 1993. "Developing a Large Lexical Database for Information Retrieval, Parsing, and Text Generation Systems." *Information Processing and Management* 29(4), 415-431.
- Covington, M. A., 1994, *Natural Language Processing for Prolog Programmers*, Prentice Hall.
- Covington, M. A., D. Nute, and A. Vellino, 1988, *Prolog Programming in Depth*, Scott, Foresman and Company.
- EasyAsk, 2001. "Company Vision." www.easyask.com.

- Ein-Dor Phillip and Israel Spiegler, 1995. "Natural Language Access to Multiple Databases: A Model and a Prototype." *Journal of Management Information Systems* 12(1), 171-197.
- ELF, 2001a. "About Us." www.elf-software.com/about.htm.
- ELF, 2001b. "Demos." www.elf-software.com/demos.htm.
- ELF, 2001c. "Demos." www.elf-software.com/store.htm.
- Epstein, S.S. "Transportable Natural Language Processing through Simplicity – The PRE System." *ACM Transaction on Office Information Systems* 3, 1985.
- Fellbaum, C., 1998, *WordNet an Electronic Lexical Database*. MIT Press, Cambridge, MA.
- Grosz, B. J., D. Applet, P. Martin, and F. Pereira, 1987, "TEAM: An experiment in the design of transportable natural-language interfaces," *Artificial Intelligence* 32(2), 173-244.
- Jacobs, P. S., ed., 1992, *Text-Based Intelligent Systems: Current Research and Practice in Information Extraction and Retrieval*. Lawrence Erlbaum, Hillsdale, NJ.
- Jagpal, Sharan, 1999, *Marketing Strategy and Uncertainty*. Oxford University Press, Oxford.
- Klein, B., R. Crawford, and A. Alchian, 1978, "Vertical Integration, Appropriable Rents, and the Competitive Contracting Process," *Journal of Law and Economics* 21, 297-326.
- Kreps, David M., and Jose A. Scheinkman, 1983, "Quantity Precommitment and Bertrand Competition Yield Cournot Outcomes," *Bell Journal of Economics* 14, 326-337.
- Language Industry Monitor, 1993, "TM/2: Tip of the Iceberg?" www.lim.nl/monitor/ibm-tm2-2.html.
- Martin, P., D. Appelt, and F. Pereira, 1983, "Transportability and Generality in a Natural-Language Interface System." Proceedings of the Eighth International Joint Conference

- on Artificial Intelligence, Karlsruhe, West Germany, 573-581, William Kaufmann, Inc., Los Altos.
- Maskin, Eric, and John Riley, 1984, "Monopoly with Incomplete Information," *Rand Journal of Economics* 15, 171-196.
- Microsoft, 2000. "English Query." www.microsoft.com/sql/evaluation/features/english.asp.
- Miller, George A, 1990. "WordNet: An On-line Lexical Database." *International Journal of Lexicography* 3(4), 235-312. <ftp://ftp.cogsci.princeton.edu/pub/wordnet/5papers.ps>.
- Miller, George A, 1995. "WordNet: a lexical database for English." *Communications of the ACM* 38(11), 39-41. <http://www.acm.org/pubs/articles/journals/cacm/1995-38-11/p39-miller/p39-miller.pdf>.
- Morrison, J., and M. Morrison, 2002. *Enhanced Guide to Oracle8i*. Course Technology, Cambridge, MA.
- Nanduri Sastry and Spencer Rugaber, 1996. "Requirements Validation via Automated Natural Language Processing." *Journal of Management Information Systems* 12(3), 9-19.
- Oi, Walter Y., 1971, "A Disneyland Dilemma: Two Part Tariffs for A Mickey Mouse Monopoly," *Quarterly Journal of Economics* 85, 77-90.
- Plane, D. R., *Quantitative Tools for Decision Making Using IFPS*. Reading, MA: Addison Wesley, 1986.
- Read, S., *The Oracle Edge: How Oracle Corporation's Take No Prisoners Strategy Has Created an \$8 Billion Software Powerhouse*. Holbrook, MA: Adams Media Corporation.
- Russell, Stuart, and Peter Norvig, 1995. *Artificial Intelligence: A Modern Approach*. Prentice Hall, Upper Saddle River, NJ.

- Salton, G., 1989, *Automatic Text Processing: The Transformation, Analysis, and Retrieval of Information by Computer*. Addison-Wesley, Reading, MA.
- Shapiro, C., and H. R. Varian, 1999. *Information Rules: A Strategic Guide to the Network Economy*. Harvard Business School Press, Boston, MA.
- Shy, Oz, 1995, *Industrial Organization, Theory and Practice*. MIT Press, Cambridge, MA.
- Shy, Oz, 2001, *The Economics of Network Industries*, Cambridge University Press, Cambridge, U.K.
- Tirole, Jean, 1988, *The Theory of Industrial Organization*. MIT Press, Cambridge, MA.
- Waltz, D.L., 1978. "An English Language Question Answering System for a Large Relational Database." *Communication of the ACM* 21(7), 526-539.
- Wang, Y.C., J. Vanderdorpe, M. Evens, 1985. "Relational Thesauri in Information Retrieval." *Journal of the American Society for Information Science*, 36(1), 15-27.
- Wilensky, R., Arens, Y., and Chin, D., 1984. "Talking to UNIX in English: An Overview of UC." *Communications of the ACM* 27(6), 574-593.
- Williamson, Oliver E., 1981, "The Modern Corporation: Origins, Evolution, Attributes," *Journal of Economic Literature* 19(4), 1537-68.
- Woods W.A., 1970. "Transition Network Grammars for Natural Language Analysis." *Communications of the ACM* 13, 591-602.
- Woods W.A., 1978. "Semantics and Quantification in Natural Language Question Answering," *Advances in Computers*, Vol. 17, M. Yovits, ed., 2-64, New York: Academic Press, Inc.