Commercialization of Natural Language Processing Technology

Deborah A. Dahl, Lewis M. Norton, and K.W. Scholz

Successful commercial deployment of natural language understanding (NLU) products requires far more than accurate technology. Although NLU technology is immature even in laboratory systems, we believe the existing technology is adequate for many commercially useful applications. Rather than describe shortcomings of the existing technology, we focus here on the supporting infrastructure necessary for its successful use. We discuss key deployment issues in both the runtime and development environments. As with any commercially deployed software, an NLU system must justify its presence in the overall system. Thus the value added by NLU at runtime must be consistent with its CPU and memory requirements. Similarly, at development time, the benefits derived from NL-enabling an application must be in line with the cost of development.

Burroughs, the Unisys predecessor, began its natural language processing research in 1981. The initial architecture of a text processing system was in place by 1986 [10] and was extended to process spoken language in 1989 [4]. Many applications were built internally to stimulate research on the basic system, and to serve as evaluation vehicles [8] and pilot systems for potential customers [2]. In addition, over 50 external sites have licensed the technology from Unisys for their own research projects.

In order to realize its strong potential in commercial applications, Unisys initiated commercialization of NLU technology in 1994. The move to commercialization revealed many requirements that were never addressed in the research environment, despite the Unisys system having been used extensively for nearly a decade in research activities. We had to address the fundamental problems of how to integrate NLU technology with other systems to provide usable solutions to real customer problems and
how to cost-effectively build and field an application. This article discusses how these
issues were addressed in the development of the Unisys product, Natural Language
Assistant (NLA).

Our focus is on applications involving NLU in the user interface. We do not deal
with applications that make use of large-scale text processing, such as machine transla-
tion, information retrieval, and data extraction. Several applications of this type were
discussed in a special issue of Communications on natural language processing [1]; here
we discuss both spoken and written applications of NLU. The spoken applications we
focus on are mainly telephone information exchanges, such as mortgage and insurance
quotations. Written applications include Web-based services located on the Internet or
an intranet.

Previous efforts to commercialize natural language technology—including English
Wizard (developed in the early 1980s as Intellect), Symantec’s Q&A, and more recent-
ly Parlance from BBN—did not achieve significant commercial success. Factors that
may have contributed to the failure of these earlier systems include:

• The limited market value of their narrow range of possible applications. Each sys-
tem interfaced to limited types of databases—Q&A to a proprietary flat database; Par-
lance and Intellect to relational databases. None of our 11 applications uses a
relational database. In fact, because NL Assistant creates an application-indepen-
dent semantic representation, no inherent limitations exist on the types of back-
end software to which it can provide an interface.
• The difficulty of developing applications using these systems. NL Assistant
addresses this issue with a set of sophisticated editors for linguistic information
and far greater built-in linguistic expertise, made possible by the advent of large-
scale linguistic resources. In addition, the availability of robust processing technol-
ogy in NL Assistant enables the system to do a reasonable job even when the
developer has not created perfect linguistic information. Although linguistic edi-
tors were available in these earlier systems, robust processing and large-scale lin-
guistic resources were not.
• Their lack of speech input capability. We find that the majority of our customers
are interested in spoken interfaces.

Current Applications

Our applications range in scope from prototypes to completed customer applications
for domains as diverse as retail banking and speech therapy. Retail banking and mort-
gage quotation applications have been developed for use with IVR (Interactive Voice
Response) systems as well as Web browsers. The Unisys product Mortgage Assistant
prompts a telephone caller to provide enough data about a desired mortgage to gen-
erate an estimated monthly payment based on the mortgage type, term, principal
amount, down payment, and so forth. Other applications include credit-card account
processing, answering frequently asked questions on the Web, and airline schedule
information.

Evaluating the cost-effectiveness of this technology is challenging due to several inter-
acting factors. The complexity of our applications varies widely, for example. Complex
dialogues with many different paths are more difficult to develop than applications that
do not require reference to a context. Spoken language systems, because of the require-
ment to develop speech recognizer grammars, are more complex to develop than text
applications. Finally, a polished commercial product is more complex than a quick
demo that is not required to be robust with input from many naive users. No simple
answers exist to the critical questions: “How well does it work?” and “How hard is it to develop an application?” In our experience, the natural language development activities—even for a full commercial product—require only about 10% of the total development time. The most time-consuming aspect of development is the design of the application itself, including specification of requirements, market analysis and dialog design. At one end of the spectrum is the Unisys Mortgage Assistant product, which required about two person-years of development, of which two person-months involved natural language development by a skilled linguist. At the other end of the spectrum, several sample text applications covering relatively circumscribed topics have been successfully developed by untrained developers with software engineering backgrounds with 2–4 weeks of development time.

Several quantitative evaluations of our applications have been performed. The fullest evaluation of a spoken language system was done for a retail banking pilot project. The application allowed users to transfer funds between accounts, check balances, and find out if a check had cleared. Overall application accuracy for 325 in-scope utterances was 89% (including speech recognition errors). Habitability, or the ability of users to stay within the confines of the system’s capabilities was good, probably due to the system’s very direct prompts. Only 3% of utterances failed due to the user going outside the system (all of these were due to the users not following instructions about numeric formats). As for a text-based system evaluation example, a mortgage counseling system by an untrained developer with one person-month of effort achieved a score of 80% correct on the first answer and 87% correct in the first two answers out of 261 utterances of unseen test data.

**Technical Issues at Development Time**

For NLU technology to be practical, application development must be fast and cost-effective. Multiple person-years of development make sense for a few highly leveragable applications, but for the majority of applications a massive development effort is not cost-justified. Application development is done with the NLA toolkit; we have addressed the goal of making development cost-effective by minimizing the required level of linguistics and programming expertise.

To minimize the linguistics knowledge required to build an application, the system includes a great deal of general knowledge of the English language at the outset, before any applications are built. This knowledge takes the form of a general English grammar, general information about English vocabulary and a general-purpose NLU engine that uses the grammar and vocabulary [3].

Natural language processing in the Unisys NLA is done by a modular, knowledge-based natural language engine (NLE) with the architecture shown in Figure 1. Processing stages include lexical lookup, syntactic parsing, semantic analysis, and pragmatic analysis. Each stage has been designed to use linguistic data such as the dictionary and grammar, which are maintained separately from the engine.

For an application of the NLE to perform acceptably, it needs information about the words used in utterances—not only a dictionary entry such as definition or part-of-speech, but information enabling the NLE to “understand” the definition and syntax. The template NLE, on which all NLE applications are based, contains such understanding-enabling information for about 3,000 English words. This core vocabulary includes a full set of prepositions, pronouns, conjunctions, and so on—the so-called “closed-class words.” It also includes a few hundred of the most frequently-used words
in the more open-ended word classes: the nouns, verbs, adjectives and adverbs. A developer can also enter information for additional words manually. Since a vocabulary of 3,000 words is insufficient for any real application, this would be a substantial task if not for our provision of linguistic servers that contain information for many more words. Information can be extracted from the servers at development time and included in NLE applications.

The four linguistic servers are briefly described as follows:

- **Lexicon Server.** The NLA lexicon server is based on Comlex, a machine-readable dictionary developed at New York University and distributed by the Linguistic Data Consortium [6]. Comlex contains detailed syntactic information for about 45,000 English words. This information, represented in a format designed for machine use, is much more comprehensive than that found in commercial dictionaries.

- **Knowledge-base server.** The NLA knowledge-base server is based on WordNet, a machine-readable hierarchical network of concepts developed and distributed by Princeton University [7]. The server also utilizes work done at the Information Sciences Institute (ISI) of the University of Southern California; ISI has supplied mnemonic names for the WordNet concepts and made them generally available to the WordNet community. These concepts correspond to real-world phenomena in terms of how people understand the meanings of words. The knowledge-base server is concerned only with concepts corresponding to nouns. There are about 60,000 of these concepts in WordNet, including ancestor concepts that provide a taxonomy to the concept set.

- **Denotations server.** The NLA denotations server, also based on WordNet and the ISI names, provides the links between words and concepts. Because many nouns have multiple senses, the denotations server has over 100,000 such links for English nouns. A word is said to denote one or more concepts, according to these links. The denotations server supplies information to the NLE enabling it to extract from the knowledge base server the concepts denoted by the words extracted from the lexicon server. Also extracted are the ancestor concepts for the denot-
ed concepts. Thus, the NLE “knows” that New York and Philadelphia are both cities, for example.

- **Semantics server.** The NLA semantics server, developed by our group at Unisys, supplies information about the semantic structure of concepts associated with English words, particularly verbs. For example, the verb “abridge” has an associated structured concept consisting of an agent doing the abridging and a theme that is being abridged. Furthermore, in an English sentence using this verb, the agent is typically found in the subject and the theme in the object. Words other than verbs can have similar information. The semantics server contains such information for over 3,700 words, mostly verbs.

Servers may occasionally be unable to supply the desired linguistic information. Information about a word may not be in the servers at development time, for example, but this is rare given the size of the servers. At runtime of a fully-developed application, the servers are not typically present, so if the end user includes a word in an input utterance that the developer did not anticipate, the information for that word will not be available. Here, the NLE can guess the required information. For instance, an unknown word will be assumed to be a proper noun, and it will be assumed to denote a dynamically created concept in the application’s knowledge base, inserted with the parent concept “thing” (since nothing else is known about the word). Another example: a known verb with no semantic information will be assigned roles such as agent or theme based on the syntax of the input utterance and statistical information about general usage of these roles in other English verbs. At development time, the developer can override such default guesses with more precise information. At runtime the default guesses are frequently sufficient for the NLE to make a usable interpretation of the input utterance.

When information from the servers is either lacking or does not reflect the application’s usage of those words, a developer chooses one of four linguistic editors to augment the server information: lexical syntactic, knowledge base, denotations, or semantic case frames. Use of these editors requires good knowledge of the English language and basic grammatical concepts such as parts of speech and synonymy, but not programming knowledge or knowledge of the system’s internal data structures. A developer would use the denotations editor to eliminate unneeded word senses extracted from the servers, and to collapse synonymous words into the same sense, thus producing a more efficient and accurate application. In a mortgage quotation system, a developer would use the semantics rule editor to give the word “mortgage” a more detailed meaning by adding roles to the concept such as “term,” “interest rate,” “points,” and “down payment” (Figure 2).

When the NLE processes an input utterance, it produces a meaning representation for that utterance, in the context of any ongoing discourse. This representation, the Integrated Discourse Representation (IDR), is application-independent. The task of an application module is to take an IDR and convert it into whatever actions perform the intended application. Such actions may be as simple as the return of a single identifier to a NLE client, or as complex as the set of steps needed to activate a back-end expert system and pass its output back to the engine’s client.

An application module is potentially difficult to implement without support from the development environment. Creation of application modules in the NLA is simplified by an extensive developers’ interface, and a modular architecture that is the basic design for all such modules. This modular architecture enables significant savings in development time. For example, when two applications have similar content (for exam-
ple, retail banking) but differ in their requirements for back-end database formatting, only the database modules need to be modified, not the modules that manage callflow or processing of the IDR. This aspect of the system is discussed in detail in [9].

Application development with the NLA Toolkit begins with collection of a training corpus for the application. The corpus can come from any source. For example in a spoken application with ASR grammars it might come from the set of utterances accepted by the ASR grammar. Each utterance in the training corpus is associated with an answer or item to be returned. The training corpus is processed by the system, which draws on the built-in linguistic knowledge provided by the grammar and servers to create an initial set of IDR's for the training corpus. For certain types of applications that do not involve dialogue management, a set of automatically generated application rules is also created. The subsequent development process primarily involves using the editors described previously to modify the linguistic information retrieved by the servers so that it is tailored appropriately to the application, and testing the rules derived from the training corpus on test data. The general philosophy differs from that used in previous systems in that the NLA development process attempts to maximize the initial contribution of the system and only involves the developer when it is necessary to correct errors, as opposed to requiring the developer to supply all the domain-specific information manually.

**Technical Issues at Runtime**

After the application has been created, it must be deployed in a cost-effective runtime environment in order to be viable in the commercial marketplace. This is particularly important for large-scale applications, such as telephone applications deployed in large call centers. Here we discuss issues we have dealt with in providing a commercially viable runtime environment.

In order for the NLU runtime system to perform the function of interpreting natural language as part of a complete system solution, it must receive language input from a source such as a speech recognizer or a remote Web browser. The result of the natural language analysis must be returned to system components responsible for callflow management or database access. Beyond this fundamental data exchange a broad spectrum of diverse configurations must be accommodated. Telephony environments ranging in size from one- or two-line installations up to major call centers receiving thousands of calls per day must be supported. Some mature existing applications with well-
developed callflow managers can be augmented by adding NLU capability as an extension to existing dialogue management capability. Other applications require the addition not only of NLU processing but also complete dialogue callflow management. Also, NL customers will frequently be committed to a particular favorite platform and operating system. We addressed these challenges by designing our runtime system with three guiding objectives:

- **Scalability**: the ability to support an arbitrarily large session count;
- **Compatibility**: the ability to NL-enable a broad spectrum of new or preexisting application domains; and
- **Versatility**: the ability to be distributed across a heterogeneous multiple platform LAN-connected environment.

We met our scalability objective by providing a runtime environment capable of managing a large number of NLEs running simultaneously, each capable of executing many simultaneous independent sessions. The multiple NLEs could be executed on any number of different LAN-connected platforms. The central component of the runtime environment is the NL Resource Manager.

As illustrated in Figure 3, the Resource Manager supervises one or more NLEs that can reside on the same platform or on any LAN-connected platform. The configuration complexity is hidden from the user by providing an API serving as a single point of contact between the NLA and the user’s environment. All elements of the NLA runtime architecture communicate over the LAN using ASCII messages that conform to the Knowledge Query and Manipulation Language (KQML) [5].

In order to provide a wide audience with access to the NLA, an Application Program Interface (NLAPI) was designed that can be called from within a customer’s application under a variety of operating systems and languages. NLAPI consists of approximately two dozen procedures that operate either by sending KQML messages to the Resource Manager or by intercepting messages it sends to the resource manager.

![Figure 3](natural-lang.png)

*Figure 3. Natural Language Processing as a system component, accessed through the NLRM.*
Manager or processing KQML messages received from the Resource Manager.

The NLAPI is coded in C with attention to portability so that it can be compiled under Win32 C and C++ as well as dialects of Unix C. Visual Basic embeddability is supported through the addition of an OCX (OLE Control Extension) wrapper. The NLAPI is packaged as an object library under Unix and a DLL and OCX under Win32.

In order to ensure compatibility with applications that vary widely in their ability to control dialogue callflow, the NLA architecture permits any NLE to execute in either of two modes. A passive mode NLE can only handle NL processing (lexical, syntactic, and semantic processing), while an active mode NLE is capable of both NL processing and dialogue callflow management. The NLA distribution includes additional essential processes for the deployment and management of a commercial system:

• **Operator Display Monitor.** A Win32 Operator Display interacts with the Resource Manager to give a system operator the current status of all platforms, NLEs, engine sessions, and other resources. The critical functionality of the Operator Display is packaged as an OCX, which can optionally be embedded in an application vendor’s user interface.

• **Transaction Logger and Log Viewer.** All messages exchanged by system components are logged by a Transaction Logger through ODBC (Microsoft’s Open Database Connectivity) into a central relational database that can be directly queried by any ODBC-compliant tools such as Visual Basic or Access. A Log Viewer initiated from the Operator Display supports routine system administration and application debugging.

• **License Manager.** To support commercial distribution of the NLA, a session license manager is built into the Resource Manager. It uses an encrypted key to limit the maximum number of simultaneously open sessions across all NLEs to the quantity purchased by the customer.

**Conclusion**

Natural language processing technology can be commercially viable. Both development time issues, such as cost and development speed, as well as runtime issues, such as scalability and platform cost must be addressed. The Unisys Natural Language Assistant represents a significant step toward widespread and practical natural language understanding capabilities for systems ranging from speech therapy to mortgage quotations.

**References**


