NLP and Industry: Transfer and Reuse of Technologies*

Leo Obrst and Krishna Nanda Jha
Boeing Defense & Space Group
Helicopters Division
Advanced Computing Technologies
P.O. Box 16858, MS P29-99
Philadelphia, PA 19142-0858
{leo.obrst, krishna.n.jha}@boeing.com

Abstract

This paper describes a useful set of NLP tools which has been successfully applied to many different kinds of industrial requirements spanning multiple domains and applications at Boeing. The tools can be combined to constitute a full-spectrum natural language system and can be customized for new domains relatively easily. To date, this array of formal and natural language processing technologies has been used to perform mass changes to legacy textual databases and to facilitate user interfacing to relational databases and software applications.

1 Introduction

Industry has many uses for NLP technology. Because the range of possible application is so varied and the practicality constraints which industry imposes sometimes quite confining, NLP components must be reusable and extendible. This paper describes a set of NLP tools which has been successfully applied to many different requirements at Boeing. The tools can be combined to constitute a full-spectrum natural language system and can be customized for new domains relatively easily. We describe the tools and a typical real application which uses them.

2 Example: Mass Change of Text

2.1 The Problem

On-line legacy databases are used daily by industry. Some of these databases consist of large amounts of relatively unconstrained texts constituting manufacturing plans and procedures, for example. These textual databases require periodic "mass changes" to correct errors and update procedures. "Mass change" means more than a simple "global search and replace" of text, in that the procedure must search for candidate structured paraphrase sets (while abstracting away from surface noise) and then apply generation rules which are context dependent on the structures. The only alternative solution to this problem is to inspect and change the texts manually, a solution which is error-fraught and expensive. The original problem can be mitigated, however, by controlling the syntax and semantics of the text prior to populating the database by using an authoring tool (for example, Boeing's "simplified English" system [17]).

In the Boeing Company, millions of manufacturing operations texts exist in legacy databases. These texts are used by an on-line planning system to stage the manufacturing of aircraft. Because there are many manufacturing process threads, with varying degrees of changeability, and many analysts and other personnel who contribute to the collection of these texts, the databases are in constant flux and contain significant noise.

When a sequence of operations must be modified, as when high volatile ozone-depleting organic compounds need to be replaced by those having low volatility, then all relevant texts must be retrieved, interpreted to understand whether they match the relevant conditions of the mass change, and then modified according to specified rules. Such a textual modification process requires robust normalization, complex pattern recognition, syntactic parsing, and semantic understanding of domain reference. Furthermore, inference is required to generate new texts based upon arbitrary change criteria.

2.2 The Application

Using formal language and NLP components, we customized a procedure to effect the mass change of on-line textual databases for the circumscribed domain of chemical treatment, prime, and finish operations. These operations (represented as texts) are performed on the shop floor in a precisely determined sequence, dependent on the aircraft design requirements and the

* This paper has benefited greatly from discussions with Gary Coen of the ACT center at Boeing, Philadelphia.

part under construction. The finish and rinse operations include applications of anodizers, primers, overcoats, and topcoats of a variety of compounds, thicknesses, and numbers of coats, to a range of treated or untreated parts of diverse material composition, and describe the manner in which the parts must be manipulated. The texts refer to these materials and processes directly (i.e., they name the materials and processes), indirectly (i.e., they name documents and standards which refer to the materials and processes), and in manners which combine direct and indirect reference. Various types of temporal and spatial information are present in the texts, including duration of finish application and drying time, and the location of areas to be finished or protected. Also present in the text are references to other documents, color codes, and miscellanous additional operations. Though circumscribed, the semantics of this domain is richly structured.

Examples of some simple plan texts from this domain are displayed below (excluding database key information):

(1) PRIME (1) COAT ZOINC CHROMATE PRIMER PER VF1.1

(2) TOUCH UP REWORK AREA ONLY APPLY (1) COAT OF BMS10-11 TYPE 1 PRIMER PER BAC5736 (F18.01) REATTACH IDENTIFICATION TAG

These examples exhibit misspellings, irregular punctuation and nomenclature, and direct, indirect, and mixed reference, which indicate the prospective usefulness of an NLP approach.

2.3 The NLP Solution: a Process View

Because these are production databases and constantly undergoing change, freezing these databases entails temporarily removing them from production use, which can be a very expensive undertaking. Hence, the automated mass-change process must be able to run reliably in a very small window of time. By distributing the processing of the texts across many Unix workstations, the time required for a typical run (ranging from 6500 to 130,000 texts) has been reduced to approximately 1.5 hours, thus minimizing downtime cost.

Figure 1 schematically represents the mass-change process. Initially, a subset of the on-line database's records are extracted and downloaded (1). The records are divided into key and text portions, made unique, and normalized (2). The plan set is then partitioned (3) according to the type of operation and/or finish material, and these partitioned sets are distributed for subsequent processing across available workstations.

Then, for each partition, the plans undergo spelling correction (4), driven by a mutual information model [1] constructed by prior exposure to and generalization over large amounts of test corpora. This process, discussed in more detail in the next section, feeds the NLP system proper. The NLP system spans the continuum from lexical tokenization (5), including the use of the two-level morphology tool PCKIMMO [2, 9, 8] which allows for a finite-state structured lexicon, through phrase structure parsing using a hybrid syntactic-semantic grammar (6), to semantic and discourse interpretation (8), and finally to the new plan generation stage (9). The tokenization and grammatical subprocesses are implemented in the C programming language. Text strings are tokenized by employing a subsystem built around lex, a Unix lexical analysis tool [1]. The grammatical processing is performed by a yacc-like LR(1) parser [1, 16] extended to include backtracking, inheritance, token-stream manipulation, and the use of semantic hierarchies, described in the next section. The semantic hierarchies (7) are also used by the later interpretation and generation modules. Most of the interpretation and generation modules are implemented in Prolog because robust inference is required. The semantic representations of those texts which fit the requirements of the change rules then undergo generation: working from the input semantic representation of an individual text and the generation rule set, a new plan is generated for each appropriate operation text. Once all texts in every partition have been fully processed, resulting in multiple sets of plans, the texts are reattached to their original keys (10) and formatted (11) to various specifications (a report to be inspected by analysts, etc.), including a database record format. The set of new database records are then uploaded to the mainframe database, and the database is again placed into production.

3 Components

This section describes in more detail key components of the NLP tool set. These include spelling correction, parsing, and semantic interpretation. The discussion of these three modules will similarly center on the mass-change application of the previous section, with additional comments on the interpretation component provided with respect to another application, that of a query interface to a project and program scheduling system. The mass change plan generation process is also described.

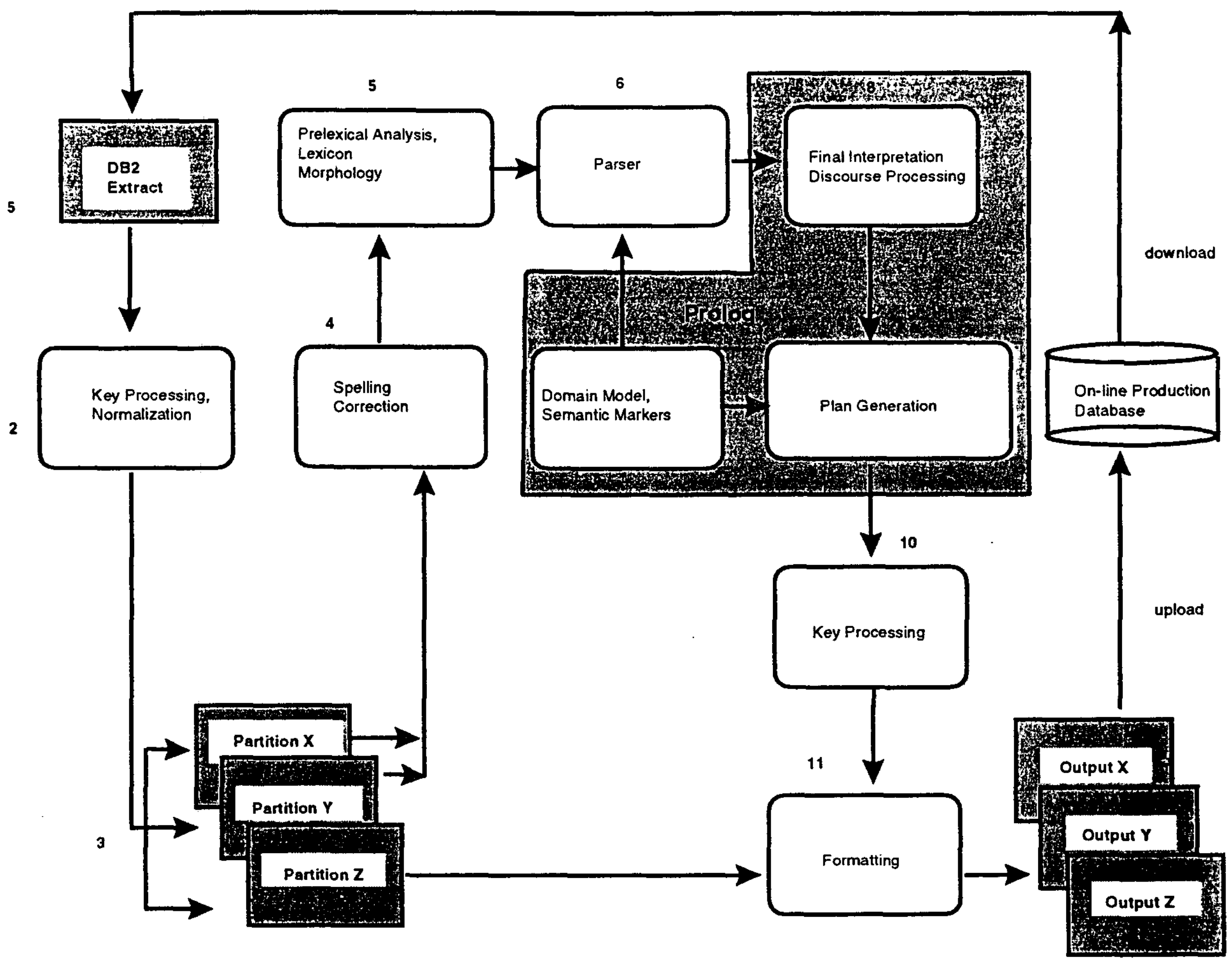
5
DB2 Extract
5
Prelexical Analysis, Lexicon Morphology
6
Parser
8
Final Interpretation Discourse Processing
Prolog
download
4
2
Key Processing, Normalization
Spelling Correction
Domain Model, Semantic Markers
Plan Generation
On-line Production Database
10
Key Processing
upload
Partition X
Partition Y
Partition Z
3
11
Formatting
Output X
Output Y
Output Z

Figure 1. Mass Change Process Flow

3.1 Spelling Correction

The spelling correction process represented by node (4) in Figure 1 utilizes a statistical *mutual information* model [5] to detect and correct spelling errors, based on the observation that spelling errors are statistically abnormal patterns. The intent therefore of spelling correction is to modify the word sequence minimally to make it statistically normal. The approach we have pursued is to use a bigram mutual information model, created by pre-processing a huge domain-specific textual corpus (obtained perhaps, as in our case, by downloading an entire textual database), to guide spelling correction over new text within that domain (Figure 2). A new model is created each time the domain changes; this is especially important if the domains are narrowly circumscribed and company-specific. In the mass change procedure, spelling correction is applied to the new corpus en masse at node (4). Statistically unlikely words are corrected to statistically likely candidates.

In general, there are problems inherent to the detection of spelling errors. For example, all unknown words encountered are not necessarily errors; they may simply not have been seen before. Furthermore, all known words are not necessarily correct; these are epitomized by typographic variations and incongruous word sequences. The mass change corpus exhibited the following occurrences (with intended word bracketed to the right):

(3) a. Typographic Variations
APPLY 2 *COSTS* OF EPOXY <*COATS*>
MARK BORE AND HOLE <*MASK*>

b. Incongruous Word Sequences
CLEAN ACRYLIC
<*CLEAR*> *FOLLOWED* WITH
<*FOLLOW*>

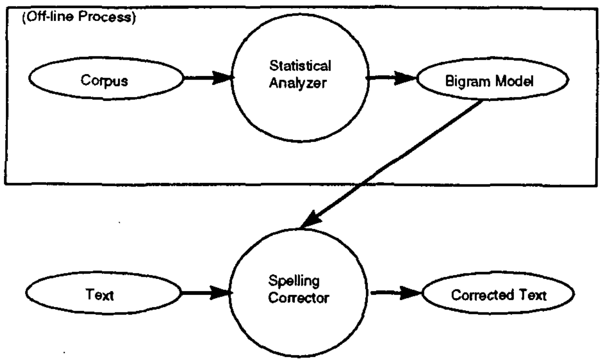

Figure 2. Spelling correction

Other anomalies which a spelling correction routine must contend with are split words (with one or more spaces intervening) and run-on words (where no space separates using bigram model two words). In addition, there is the possibility that the error-to-correction mapping is non-invariant.

A statistical approach to spelling correction has some advantages and some disadvantages. Among the advantages are: it corrects the majority of errors, those classified as nonwords, misspelled words, word-splits, and run-ons; the automated acquisition of domain-specific data is easily maintainable; and the use of a statistical model enforces consistent lexical usage. A disadvantage is that correlated recall and precision may not be high, i.e. some errors may be missed and some may be corrected incorrectly. However, reasonably good recall (>75%) coupled with very high accuracy (>95%) can be expected. Other disadvantages are: there is no clear strategy for multi-error detection and correction, and the fact that such a large corpus (20 megabytes in our mass change corpus) is required to create a good statistical model.

3.2 Parsing

For parsing, we use a generalized LR(1) shift/reduce parser [16, 1, 10]. Like yacc (which, given a grammar, generates a parser for that grammar), our parser precompiles the CFG grammar into a state-transition table. The parser exercises CFG grammar rules annotated with syntactic and semantic action routines, thus allowing for synthesized and inherited attributes. In addition to the rules, other knowledge stores integrated into the parser's processing are a thematic role hierarchy and a semantic domain network, both of which are also used by lexical entries in a morphologically partitioned lexicon. The parser uses a linked list of structured tokens (displayed in 4 below), and returns only one parse. To facilitate robust parsing, the parser also allows the developer to activate grammar-directed token dropping, token hypothesizing, and token type coercion.

(4) Token Structure

<*id*:	numerical identifier for token
surfform:	surface form (i.e. actual string) for the token
rootform:	root form of the token
value:	value (semrep) associated with id
assertions:	*[]*
scat:	subcategorization requirement for the token, where the scat format is (ext_arg int_arg1 int_arg2 ...), and where each argument must be a grammar symbol (exception: int_arg1 may be a string enclosed within # e.g. #into#contact#with#); ext_arg may be NULL/nil;
feature:	feature associated with token
next:	ptr to next polysemous token>

The parser permits arbitrary backtracking, including that over polysemous or composed tokens (idioms), over grammar rules, and over object hierarchies (entity, property, and predicate types in the hybrid domain model), though in practice time and node limits are set. The backtracking facility also includes the developer-specified *cut*, an operator to force the termination of a grammar rule. An example of backtracking over polysemous tokens is displayed in the following abbreviated trace from the mass-change process. As noted, we employ a hybrid syntactic-semantic grammar, primarily because such a hybrid permits generality (at higher nonterminals) and specificity (at terminals and lower nonterminals).

(5) Backtracking over Polysemous Tokens

Lexicon (abbreviated):
```
FINISH    : FINISH_VERB
          : FINISH_NOUN             (isa
MATERIAL)
```

Parsing:
```
FINISH 1 COAT OF FINISH
mismatched  string  SCAT  [required:
code] / [found: 1[1]]
....
Difficulty in parsing: no transition
for token NUMBER[168] from state 83
current  stack  (in  reverse):  state-
stack[1]: [FINISH[124] ]
backtrack ...
nbar : ENTITY
a_nbar : nbar
a_nbar : a_nbar NUMBER
nphrase : a_nbar
Difficulty in parsing: no transition
for token COMPOSITION[69] from state
75
current  stack  (in  reverse):  state-
stack[2]: [COATING[62] nphrase[4109]
]
....
```

```
current stack (in reverse): state-
stack[4]:                [FINISH[124]
COMPOSITION[69]          fin_qfr[4126]
FINISH_VERB[126] ]
backtrack ...
fin_n : MATERIAL
fin_np : fin_n
of_np : COMPOSITION fin_np
fin_np : fin_qfr of_np
s_imp : FINISH_VERB fin_np
sentence : s_imp
discourse : sentence
top : discourse EOS
```

When enabled, the token-dropping option allows a grammar rule to be matched by dropping a token (from a pre-specified set of droppable tokens), and is only applied when a sentence will not parse without dropping the token. In addition, the parser will also hypothesize a token when the input sentence will not parse strictly by using the grammar rules. Similarly, the parser will coerce the unexpected type of a token to a type which is acceptable, should the parse otherwise fail.

3.3 Semantic Interpretation

The mass-change procedure does not require the complex referential semantics that NLIs require. The semantics and the discourse components can be simpler because the application requirements are simpler. In all our NLP applications, however, both domain-dependent and -independent information constitute the semantic model, which is jointly used by the grammatical module (written in C) and the interpretation/generation module (written in Prolog). Each token has a semantic marker which acts as an index into the semantic domain model. The morphologically generative lexicon is the primary knowledge store associating the input (surface) text form, its tokenization, and the semantic marker. The grammatical module uses the lexicon to drive its work, but also uses the semantic model directly to enable type inheritance and, in some cases, the type coercion of semantic markers.

The semantic domain model consists of a set of assertions of the form

object(Child, [Relation, Parent])

where Relation is either 'isa' or 'ispart', and the three possible roots of the hierarchies are 'entity', 'predicate', and 'property'. These are defined by a developer and entered into a the GraphEd tool [14], a graph editor which outputs an ascii representation of a network. The ascii form can be transformed and used by both the parser and the backend Prolog interpretation processes.

The output of the grammatical module is a combined syntactic-semantic representation of the input plan text in the form of a list of binary predicates capturing the tree structure. Each predicate is of the form:

predicate(skolem-constant, value)

with skolem constants representing the nodes of the tree. The semantic entity markers are those items which are the values of 'instance' predicates, as

instance(n9, person)

asserts that 'n9' is an 'instance' of semantic class 'person'. The syntactic-semantic representation is then asserted as the primary knowledge store in the final interpretation and generation module.

Additional knowledge sources used in the Prolog interpretation and generation module are: a database of finish codes and their associated information, including the number of coats of application required, color number, color name, and material type of each relevant finish code; a set of material-specific databases which include the materials and the associated generation requirements rules; and a task-driven tree-walker that traverses the semantic representation of a plan to extract information requested by the generator.

3.4 Plan Generation

The text plan generator directly executes rules representing the output requirements of the new plans. Prior to executing these rules, however, the generator determines whether the original input plan is well-formed, valid, and consistent. Then, using the domain model, the finish code and material databases, the requirements rules, and the semantic tree-walker, the generator creates new plans.

In other cases, the generator detects that a meta-constraint such as "Only one operation should exist per plan text" is violated. It flags the text as anomalous, indicating the constraint violation, but still tries to generate a reasonable output text. A post-generation process diverts constraint violations to a separate stream which results eventually in the creation of a special report. Texts which violate constraints are not changed and uploaded; instead, these are evaluated by a human domain expert, who adjudicates the suggested changes individually. For example, in (6) the original plan consists of multiple run-on sentences with no punctuation. The NLP system determines that there are actually three sentences, two of which refer to application of finishes. With this information, the generator determines that one of its meta-constraints has been violated, generates its best guess at an output text, and then annotates that text with the constraint violation message.

(6) Example of Generated Text

Input:

TOUCH UP REWORK AREA ONLY APPLY (1) COAT OF BMS10-11 TYPE 1 PRIMER PER BAC5736 (F18.01) REATTACH IDENTIFICATION TAG

Generated Text:

<<FOLLOWUP: MULTIPLE OPERATIONS SPECIFIED.>>
TOUCH-UP FINISH REWORK AREA ONLY AS/IF REQUIRED PER ENG. DWG. PRIME PER F-18.01. REATTACH IDENTIFICATION TAG.

3.5 Interpretation and Other Applications

The mass-change application is fairly simple. More complicated NLP applications require ellipsis and pronominal resolution, and more richer referential semantics. An NLI to a relational database, for example, requires an explicit recursive semantic composition process. This is why our deeper semantics in Prolog closely parallels that which a categorial analysis would furnish, i.e., using function application and composition over lambda forms, per treatments such as [11, 12] and using a semantic theory such as DRT [7]. Such an approach allows one to compose a semantics in a principled manner and to interpret with respect to the domain model. Nevertheless, to this point, in an interface to a project and program scheduling system, we have attempted only to render semantics for scope-underspecified quantifiers, negation, and numerical and temporal constraints. Tense and aspect (e.g., [15]), distinctions among plural readings of noun phrases, and a deeper lexical semantics, have so far not been elaborated, but are planned. In [3], e.g., a lexical semantics based on [6] will be developed. Finally, a tool like [4]'s Prolog-to-SQL compiler can prove useful for mapping the final referential semantics to a specific database or domain model.

4 Conclusion

The NLP tools described in this paper have been used a number of times to effect the mass-change of on-line textual databases. The cost savings over other methods has been significant (we estimate, for example, that in four years, 20,000 man-hours have been saved over manual methods). By representing core semantic components in Prolog, we expect to minimize the work needed to accommodate radical domain changes in the future, though application-specific manual work must still be performed to update the lexicon, modify the grammar, and elaborate new referential semantics. These same NLP tools, modified to accommodate primarily lexical differences, a more complicated semantic domain model, and deeper interpretation, have been employed in building NLIs to legacy databases and applications, in a resource-conserving manner.

References

[1] Aho, A.; Sethi, R.; and Ullman, J. (1986). Compilers: Principles, Techniques and Tools. Reading, MA: Addison-Wesley

[2] Antworth, Evan L. 1990. PCKIMMO: A Two-Level Processor for Morphological Analysis. Dallas, TX: Summer Institute of Linguistics.

[3] Barrett, Tom; Coen, Gary; Hirsh, Joel; Obrst, Leo; Spering, Judith; Trainer, Asa. 1997. MADEsmart: An Integrated Design Environment. Submitted to 1997 ASME Design for Manufacturing Symposium.

[4] Draxler, Christoph. 1993. A Powerful Prolog to SQL Compiler. CIS Centre for Information and Language Processing. Ludwig-Maximilians-Universität, München, Germany. August 16, 1993.

[5] Dunning, Ted (1993). Accurate Methods for the Statistics of Surprise and Coincidence. Computational Linguistics19:1, pp. 61-74.

[6] Jackendoff, R.S. 1990. Semantic Structures. Cambridge, MA: MIT Press.

[7] Kamp, Hans; Reyle, Uwe. 1993. From Discourse to Logic: Introduction to Modeltheoretic Semantics of Natural Language, Formal Logic, and Discourse Representation Theory. Dordrecht: Kluwer Academic.

[8] Kartunnen, Lauri. 1983. KIMMO: A General Morphological Processor. Texas Linguistic Forum 22: 163-186. University of Texas, Austin, TX.

[9] Koskeniemi, Kimmo. 1983. Two-Level Morphology: A General Computational Model for Word-Form Recognition and Production. Publication No. 11. Helsinki: University of Helsinki Department of General Linguistics.

[10] Marcus, Mitchell. 1980. A Theory of Syntactic Recognition for Natural Language. New York: McGraw-Hill.

[11] Moortgat, Michael. 1988. Categorial Investigations: Logical and Linguistic Aspects of

the Lambek Calculus. Foris Publications, Dordrecht, Holland.

[12] Morrill, Glyn. 1994. Type Logical Grammar. Dordrecht: Kluwer Academic.

[13] Obrst, Leo; Nanda Jha, Krishna; Coen, Gary. 1996. Mass Change of On-line Textual Databases Using Natural Language Processing. Industrial Applications of Prolog Conference and Symposium (INAP-96), Tokyo, Japan.

[14] Paulisch, Francis Newbery. 1993. The Design of an Extendible Graph Editor. Lecture Notes in Computer Science 704. Berlin, Heidelberg, New York: Springer-Verlag.

[15] Verkuyl, Henk J. 1996. A Theory of Aspectuality: The Interaction Between Temporal and Atemporal Structure. Cambridge: Cambridge University Press.

[16] Tomita, Masaru (1985). Efficient Parsing for Natural Language. Dordrecht: Kluwer Academic Publishers.

[17] Wojcik, Richard; Harrison, Philip; Bremer, John. 1993. Using Bracketed Parses to Evaluate a Grammar Checking Application. Proceedings of the 1993 ACL Conference.