Description of the American University in Cairo's System Used for MUC-7

Christian R. Huyck

The American University in Cairo

c.huyck@mdx.ac.uk

INTRODUCTION AND BACKGROUND

Portions of the American University in Cairo's MUC-7 system, MUC7-Plink, have participated in every Message Understanding Competition since MUC-4. The *Plink* parser was developed at the University of Michigan where it formed the core of the systems entered in MUC-4 [2] and MUC-5 [1]. Recently, the Plink parser was added to GATE [6] to facilitate interaction between language processing modules. Most of the modules used in MUC7-Plink were already in GATE having been imported from the LaSIE system used in MUC-6 [8].

GATE provides an environment that greatly simplifies the reuse of existing natural language models. When the call for participation in MUC-7 was made, I was a faculty member at the American University in Cairo, and had several students who were considering participating along with me in MUC-7. I could have easily divided the tasks and had, for instance, one student work on the Gazetteer, one work on coreference and perhaps a small group work on discourse interpretation. Along with the existing Plink parser this would have comprised a largely new system. Unfortunately, I left Cairo, and had only a very small amount of time to develop the system. Furthermore, I had to develop the system at home on my PC. Fortunately, GATE already had all of the modules that I needed, and ran (albeit slowly) on my PC. I did have to modify some things, but with a very small amount of effort, I developed a working MUC-7 system.

Sadly, due to the lack of resources, the results of the system were poor, and by no means reflect the ceiling of the technology. They do however show how easy it is to perform relatively well with virtually no development time.

The MUC7-Plink system is largely that of the Sheffield system. It differs largely by the use of an on-line lexicon, and the use of a different parser. This parser was used for the University of Michigan's MUC-5 entry, but the grammar and parsing heuristics have been rewritten to take advantage of the on-line lexicon, the Gazetteer, and the automated part of speech tagger. The parser also produces significantly different output for the XI discourse interpreter [17].

In the rest of this document I will first describe the system; this will include a module by module description of the components, and a brief description of GATE. I will then describe the performance of the system; this will include a summary of MUC7-Plink's scores on the TE, TR and ST tasks, a brief summary of how development time was spent, and a walk through of the sample article. I will conclude with a few observations.

SYSTEM DESCRIPTION

Architecture Overview

The MUC7-Plink system was composed of ten modules which were run in succession on each text. In order, they were:

- Tokenizer
- Sentence Splitter
- Tagger (Brill)
- Gazetteer
- Morphological Analyzer
- On-Line Lexicon
- Plink Parser
- Name Matcher
- Discourse Interpreter
- Template Writer

Eight of these modules were used in the Sheffield MUC-7 entry. The only ones that were substantially different were the Lexicon and the Parser. I will briefly summarize the others and give a more expansive description of the Lexicon and the Plink parser.

tokenizer

The tokenizer reads the input stream and segments it into small chunks that are roughly equivalent to words. It is an executable file compiled from a C program; the C program is generated from a Lex [10] input file. The token is the most commonly used unit of data for processing in GATE, and the tokenizer guarantees a somewhat uniform representation. These tokens are added to the GATE database; a separate database is maintained for each document. Additionally the tokenizer adds section annotations to mark areas of the text.

Each of the GATE annotations have a start and end byte which define a span. The span specifies the offset in the document to which the annotation applies. So, the token associated with "Ford" might have the span of byte 152 to 156, while a section might have the span of 0 and 562. Additional annotation specific information might be added to each annotation.

sentence splitter

The sentence splitter is a perl script which notes the sentence boundaries. These boundaries are added as annotations into the GATE database; the annotation includes the offset of the sentence in the document (the span) and all of the tokens which are constituents of the sentence.

tagger

The Brill tagger [3] is a part of speech tagger that has been extensively trained on Wall Street Journal Text. It annotates tokens with their part of speech. Since an annotation already exists

for each token, more information is simply added to each token annotation thus consolidating information.

These parts of speech are not entirely compatible with the results of the Gazetteer or the Lexicon. These conflicts are resolved before parsing begins.

gazetteer
The majority of nominal semantics in the system comes from the Gazetteer. It is a Lex [10] based system of 44 lists. Each list represents a different semantic category. The lists include companies, airlines, aircraft manufacturers, cities, provinces, titles, first names, bodies of water and aircraft names among many other things. There are about 200,000 bytes of text in the lists making roughly 10,000 entries. The system is relatively easy to modify. Addition of new elements to the list is simple, and the addition of a new list is also simple.

In addition to lists of proper names, some are lists of key words that signal certain semantic categories. For instance there is a list of organization signal words such as University, Hospital and Laboratory. These words alone are not sufficient to mark an organization, but if they occur next to an unknown proper noun they suggest that that proper noun is an organization. This adjacency, and thus categorization, is noted in the parser.

The Gazetteer was largely the one used in the Sheffield system. However, near the end of development, I had to freeze the lists while they were slightly modified at Sheffield. In general this did not matter, but a large number of launch event specific changes were not completely incorporated. The largest problem here was that spacecraft were not incorporated. Since rockets were needed for the scenario template task, virtually no scenario templates were generated.

GATE is not a strictly linear system. Module A must be run before Module B when B needs information from A. However, if neither is dependent on the other they can be run independently. [1] Since the Gazetteer does not depend on morphological analysis, it could be run before or after the tagger.

morphological analyzer
The morphological analyzer takes all nouns and verbs and returns the root form and the suffix. The root form is often used as a semantic primitive. So the semantics for "report" is the same as the semantics for "reports" or "reporting". The analysis is done by some regular expression rules and a list of several thousand irregular exceptions derived from the exception list used in Wordnet [16].

lexicon
The version of Plink used for MUC-4 and MUC-5 had a hand-crafted lexicon. Each lexical entry

[1] Theoretically, independent modules can be run in parallel, but the current GATE system does not implement this feature.

was a complex feature structure, and was rather difficult to construct. Words that were not specifically in the lexicon were assumed to be proper nouns of no particular semantic category. It would be more effective if an on-line lexicon could be used to reduce the work load because the lexicon would both ease transition to a new domain, and reduce the time need to maintain Plink's own lexicon.

Longman's Dictionary of Contemporary English (LDOCE) [9] has electronic versions. One of these versions was selected, and added to GATE. The desired word (root form) was passed to LDOCE and it returned the definitions of the word that it found. Each of these definitions were added as there own tokens to the GATE database, with spans that corresponded to the token.

Initially, each definition of the word was left in as an annotation. Plink was allowed to choose between the definitions. Unfortunately, on medium size documents the large number of lexical entries tended to slow my machine down due to memory limitations. This meant that some pruning had to be done before addition to the GATE database.

The Plink grammar that I developed roughly follows the HPSG [14] formalism. This requires rather sophisticated lexical entries. The addition of LDOCE has enabled me to begin to develop a more complex lexical system. Eventually, these definitions will include semantic and complex syntactic features which should enable more effective parsing, and more useful semantic results which can be passed along to discourse analysis.

The version of LDOCE that I used has semantics and selectional restrictions, but they seem to be inconsistently entered. Thus the information gathered from LDOCE is currently not very useful.

plink

The PLINK parser was designed for the fifth Message Understanding Competition (ARPA-93). PLINK does full fledged parsing creating exactly one syntactic-semantic representation of a given sentence. Additionally, PLINK parses in linear time thus speeding parsing. PLINK is closely related to the Marcus parser [13] using a stack of constituents. Plink uses a heuristic rule selection mechanism based on the contents of the stack to select which grammar rule to apply at each step. These heuristics have access to elements of the partially completed parse and select the rules based on a preference mechanism. The preferential mechanism is based on a small number of rankings (currently 6), so the system can select several rules and rank them.

PLINK uses a standard-unification based grammar or UBG [15], and is derived from the LINK parser [11]. The use of a UBG enables PLINK to encode grammar rules that have both syntactic and semantic components. Since the parser has access to syntax and semantics, it can take advantage of both types of knowledge to make parsing decisions. This allows parsing to proceed in one-pass and eliminate a great deal of ambiguity. PLINK also includes an inheritance hierarchy of semantic components. A more thorough discussion of PLINK and the MUC-5 system can be found in [7].

The grammar that was used was hand-crafted. Though it does not adhere to any specific linguistic theory, it is similar to the HPSG grammar of Pollard and Sag [14]. The grammar rules are quite standard except in many cases they are more amenable to one-pass parsing. For instance left-recursion is avoided. These rules still recognize the same language, but some gram-

matical manipulation improves one-pass parsing. Rules to handle agrammatical phenomenon were derived with HPSG in mind, though of course, they differ from standard HPSG rules.

The parsing model is based around a stack and selection rules. The stack is a standard parsing stack. Constituents were added to the stack, and when appropriate a grammar rule was applied to the stack modifying the top elements of the stack. I tried to keep the stack small, and in earlier experiments the stack never exceeded a size of seven constituents when it was parsing grammatical phenomena.

At any given time a number of actions could take place. A new element could be pushed onto the stack or one of a number of grammar rules could be applied. Selection rules were used to choose the next action. Like the grammar rules themselves, the selection rules are themselves UBG rules. The selection rules inspect the stack, and give a preference weighting to each of the valid options. For example:

...
det
noun
period

Example 1. A Sample Stack

Here "..." represents other elements lower on the stack and period (the punctuation mark) represents the most recently added element. All of the selection rules are unified with the stack and (for the sake of example) two selection rules match.

```
(Selection-Rule 1                (Selection-Rule 1
  (good NP-from-det-noun)          (best abbrev-eats-period)
((1) = det)                      ((1) = noun
  (2) = noun                       (2) = period))
  (3) = nil))
```

Example 2. Selection Rules that Match the same stack.

If the NP-from-det-noun rule was applied the stack would be changed to

...
NP
period

Example 3. Stack If the First Rule is Successfully Applied

If the abbrev-eats-period (abbreviation absorbs a period) rule was applied the stack would be changed to:

...
NP
period

Example 4. Stack If the Second Rule is Successfully Applied

Which of the two rules is actually selected? Grammar rules are selected based on a preference ranking. In the current system the ranking is best, good, fair, last, spec-agram and gen-agram. The best rule is applied first. When the stack is as it is in example 1, the abbrev-eats-period rule is applied first. If it succeeds a new round of rule selection begins. If it fails, then rules from the next level, in this case NP-from-det-noun are applied. This continues until all rules fail. If multiple grammar rules are selected with the same preference ranking, then they are ordered randomly.

If no rule succeeds a new constituent is pushed onto the stack. This is could be implemented by the selection rule:
(Selection-Rule 1 (gen-agram push))

This rule always succeeds and the keyword push is used to push a constituent onto the stack. Other selection rules may take advantage of the push mechanism, when more lexical information is needed to make a parsing decision.

This parsing mechanism allows no backtracking. Consequently, this assures that the parse occurs in linear time. There is evidence that humans backtrack when parsing [4], [5]. In this sense PLINK is not a full-fledged model of human parsing.

In example 2, I actually specified the names NP-from-det-noun and abbrev-eats-period. This is the actual name of the grammar rule; that is the selection rules actually encode the grammar rule by name. The name of the grammar rule is specified in the grammar rules (pref name) feature. The grammar rule for NP-from-det-noun looks like example 5.

(Grammar-Rule NP
(((1) = det
  (2) = N
  (2 head syn type) = common
  (pref name) = NP-from-det-noun)))

Example 5. A Sample Grammar Rule

The MUC-7 domain is an open ended domain of newspaper articles. These articles often have grammatical and spelling errors. Furthermore, the lexical mechanisms are not always correct. For example, occasionally words are mis-tagged. Consequently, the domain is ideal for robust parsing techniques. The simple technique that PLINK uses for robustness is low ranked rules. High priority rules handle grammatical and specific phenomena; medium priority rules handle grammatical and general phenomena; low priority rules handle agrammatical phenomenon.

A working version of the Plink parser existed by the time of the dry run. The parser was in GATE, and was receiving input from earlier modules via the GATE database. However, the grammar was designed to recognize general noun phrases. Some modifications had to be made to generate the appropriate semantic category. For instance, the parser might encounter "Robert R. Smith". This would be correctly recognized as an NP, but it would not state that it was a person. For the purposes of all of the MUC tasks, this information was needed. Consequently, new grammar rules had to be added. Since Robert is in the Gazetteer, the semantic type of "Robert" would be person and an NP formed from it would also be person. However, the type of "R." and "Smith" would be unknown. Thus a grammar rule Example 6. was needed.

```
(Grammar-Rule NP
(((1) = NP
  (2) = N
  (1 head sem) = person
  (pref name) = ng-from-NGperson-N
  (head sem) = person)))
```

Example 6. A Semantically Specific Grammar Rule

Example 6. of course conflicted with an already existing grammar rule which took the exact same constituents, but took the semantics from the second noun. A higher ranking parsing heuristic was made for the *ng-from-NGperson-N* grammar rule and it was always selected first. It only succeeded when the semantics were correct, so non-person NPs were unaffected.

A total of 11 grammar rules, and 13 selectional rules were added for the MUC task. All of these were developed during the training phase and were thus specialized for the aircraft accident domain. It would be valid to say that this was the only work done on MUC7-Plink for MUC-7. These rules were written in a few hours over several afternoons. One of the advantages of the Plink approach is the simple integration of domain specific grammar rules.

The main modifications from the MUC-5 system were a new grammar for a new tag set, and the introduction of lazy unification to speed heuristic rule selection. The new grammar was needed since the tag set had changed. The MUC-5 tag set was specific to our hand-crafted lexicon. It now uses a combination of the tags used by the Brill tagger, the Gazetteer, and LDOCE. This has been combined with a hierarchy of syntactic classes, to enable more general rules to be written. For example, instead of one syntactic class for comma, and one for each of the other punctuations, I have combined this into symbol, but each symbol has a head feature which is the symbol. A general rule can be written to look at the lexical class 'sym', or a specific

rule can be written to look at the lexical class 'sym' which has a head feature dollar for the dollar sign.

Lazy unification is now used during rule selection. In the MUC-5 system full unification was used, and this lead to large structures being built unnecessarily. A future improvement would introduce lazy unification into grammar rule application. There is evidence that this would further improve parsing performance [12].

Finally, a great deal of modification was needed to produce the correct input for the XI discourse interpreter. Fortunately, this was mostly a matter of post-processing. Plink standardly produces a list of verb frames. XI wants a list of quasi-logical predicates. It is relatively simple to change the frames into predicates. However, the XI system that was used needed a certain set of predicates. A large amount of work was needed to assure that the correct predicates were being produced. This is where the majority of work for MUC7-Plink happened. What was produced was a list of entities and relations between entities. The entities could be based on nouns or on verbs.

name matcher

This is a C++ program used as part of the coreference mechanism. If a name, or part of name, occurs in the list of entities, they are combined into one entity. This is a useful preprocessing step for the Discourse Interpreter.

discourse interpreter

The discourse interpreter was developed using the XI knowledge representation language [17]. The input to the interpreter was a series of entities and relations between entities. The interpreter had rules which built new relations and reclassified the entities. One particular important set of entities and relations was the MUC-7 specific Element, Relations and Scenarios.

The only work done for MUC7-Plink was to produce the appropriate input for the discourse interpreter. Unfortunately, this work was incomplete, particularly for the final test domain. This lead to very low recall measures in all three tasks.

An additional problem was that the coreference mechanism, which was largely implemented in the discourse interpreter, assumed that entities had a particular property. However, this relation was added by the Plink parser. This lead to a reduction in precision particularly in the Template Element task because entities that corefered in reality were not associated by discourse interpretation.

template writer

The template writer is a prolog program that simply scans through the discourse model. It looks for certain types of entities and relations, formats the information for them in an appropriate manner, and generates the templates which are the results of the system.

General Architecture for Text Engineering

This whole system was developed as a system of the General Architecture for Text Engineering or GATE [6]. Text processing modules are added to GATE, and these modules can be combined

into a system. Once modules are added they can be combined in different ways to form new systems.

GATE provides a Tipster compatible database mechanism. The database store is organized around documents. Each document has its own set of annotations. Modules take input from the database, process the input, and generate output which is then usually placed into the database.

The simplest way to add a new module to GATE is by writing a wrapper that interacts directly with the database. The wrapper gets annotations from the database and writes it to a file; the code for the module is then called with the file as input. It then produces an output file which is read by the wrapper and put into the database. Some modules, such as the name matcher, do not communicate this way. However, integrating a module in this fashion is not very difficult, and it allows the module to run without GATE if an input file exists.

GATE currently has about 40 modules with complete wrappers. Addition of a new module varies in complexity, but can be done in well under an hour for simple systems, and in 2 days for complex systems such as the ANLT parser. Since processing can be independent of GATE, the source language of the new module is irrelevant. MUC7-Plink has modules written in C, C derived from Lex, C++, Lisp, Perl and Prolog.

SYSTEM PERFORMANCE

Scores

	Recall	Precision	P&R
ST	1	43	1.45
TR	14	75	23.66
TE	36	68	47.40

Table 1. System Results

MUC7-Plink generated scores for the Scenario task, the Template Relations task and the Template Element task. The scores were lower than expected, but not much lower. No development was done on the Launch Event domain. A small amount of work could have raised the P&R scores to 20 for ST, 40 for TR, and 60 for TE; these are roughly the scores on the tasks in the Aircraft Accident domain on texts that were run blindly. Of course a reasonable amount of work on the system could have raised the scores much higher.

Development Time

The only way that MUC7-Plink excelled for the MUC-7 competition was development time. No time was spent on the Launch Event domain, and very little time was spent on the Aircraft Accident domain. A summary of the time spent in development is below.

- 15 hours development on Aircraft Accidents
- 0 hours development on Launch Events

- 48 hours on integration into the GATE/LaSIE Discourse Interpreter
- 80 hours spent adding Plink and LDOCE to GATE
- 90 hours running the final test

There was no time spent on development in the final test domain. The TE and TR scores are reasonable because some time was spent in development on the similar training domain of Aircraft Accidents. 48 hours was spent on modifying the output of the Plink parser to fit with the XI discourse interpreter that was used. This could reasonably be considered part of the MUC-7 effort. Roughly 80 hours were spent in adding Plink and LDOCE to GATE, in the summer of 1996. The integration process has been improved since then, and adding two similar modules would probably take under 40 hours effort.

The majority of the time was spent on running the final tests. I was running on a PC-586 at 90 MHz, with 16 Meg of RAM. This lead to very slow processing. The longest article took over 6 hours to process. An average article to 30 minutes to process up to the discourse interpreter. 10 minutes was spent on parsing, and 5 minutes was spent on lexical lookup. Roughly 10 minutes of the remaining time was spent interfacing with the GATE database. This is clearly a weakness of the GATE model and needs to be improved.

Discourse analysis was taking much too long, and there would have been no way to run all of the texts on my PC. Fortunately the GATE approach of reading from the database, writing to a file and then calling the module was very helpful; it enabled me to write input files for the discourse interpreter, ftp them to a Sun workstation and run them there. Roughly half of the texts were run this way, and almost all of the texts over 4000 bytes.

The major problem with this long running time was that it left no time for development on the Launch Event domain. An overnight run of texts would have enabled development of the MUC7-Plink system to have much higher results. Still it is quite remarkable that one can enter MUC-7 on a system almost solely run and developed on a low-end PC.

Walkthrough

I will concentrate on the sentence *"The China Great Wall Industry Corp. provided the Long March 3B rocket for today's failed launch of a satellite built by Loral Corp. of New York for Intelsat. "*

The tokenizer reads in the text and adds annotations like:
206 token 1118 1121
207 token 1122 1127
for the words *The* and *China*. 206 refers to the annotation number, and 1118 and 1121 is the span of the token in the text.

The sentence splitter divided the document into sentences including the above sentence as the annotation:
1139 sentence 1118 1275 constituents: 206 207 ....
This annotation says it is a sentence that goes from 1118 to 1275 and has the tokens 206, 207 etc.

The tagger modifies the token annotations by adding part of speech information.
206 token 1118 1121 (pos: DT)
207 token 1122 1127 (pos: NNP)

The Gazetteer looks up words and finds among others:
5007 Lookup 1253 1259 (tag: location) (type:city)
5008 Lookup 1244 1249 (tag: organization) (type:company)
for *New York*, and *Loral* respectively. Note that *New York* does span two tokens, thus Lookup can not be directly associated with tokens in the database.

The morphological analyzer adds root and suffix annotations to verbs and nouns.
231 token 1235 1240 (pos: VBD) (root: build) (affix: ed)
232 token 1244 1249 (pos: NNP) (root: loral) (affix:)
are the annotations for *built* and *Loral*.

LDOCE looks up words and adds rather complex annotations. An example is:
6428 ldoce_entry 1172 1177 (homograph: 0) (sense: 0) (part_of_speech: -)
(grammar_info: -) (subject_code: -) (case_info: -)
for the word March. As noted this information is not currently very useful but slots are left open for a more effective lexical retrieval mechanism.

The Plink Parser is then run on the sentence and generates a syntactic structure for the sentence, which we will ignore, and a semantic structure for the sentence. The annotation is:
7867 semantics 1118 1275 (qlf: [fail(e251), lobj(e251,e252), launch(e252) ....])

The quasi-logical forms that are of interest are: *organization(e256), name(e256, offset(1244, 1255)), city(e257), name(e257, 'new york'), apposed(e256,e257), of(e256,e257)* This says that *Loral Corp.* is an organization which has an of relation with the city *New York*.

In this particular text, the name matcher finds no matches.

The discourse interpreter finds an of relation between an organization and a location. The interpreter has a rule that adds a location_of predicate if this relation holds so a new predicate *location_of(e256,e257)* is added. The discourse interpreter in turn writes information back to the database. An example is:
8132 xi_instance 1118 1275 (class: e7 ¡– city(_))) (props: location_of(e6,e7),
country(e7, 'United States'), of(e6,e7)...)

The template writer reads these xi_instance annotations and prints the appropriate template elements and relations for in this case, *Loral Corp.* and *New York*.

OBSERVATIONS

GATE made MUC7-Plink possible. Without GATE it would have been impossible for me to develop a system capable of participating in MUC in under a few weeks of work. GATE does

have some weaknesses: adding a new module to GATE while simple is not transparent; accessing the database is quite slow. However, it has been a very useful development environment.

Plink has also shown to be quite useful. It was quite easy to add new rules for a new domain to Plink. The end result of parsing is easily translated into the quasi-logical form needed by the discourse interpreter. This comes from it being a full-parser which generates one interpretation, and generates a full semantic interpretation along with a syntactic one.

MUC7-Plink can be most usefully seen as an example of how to build a system that can very easily be moved to a new domain. Assuming a working system, for say the MUC-6 Succession Event task, three main modules need to be modified: the Gazetteer, the Parser and the Discourse Interpreter. Using the modules in MUC7-Plink only domain specific data needs to be changed and the actual programs remain constant.

The Gazetteer needed several lists changed. The parser needed to add several grammar rules, and for Plink selection rules, to account for the lists. Switching to a new domain would again call for new lists and new grammar rules. However, this data is all based around Noun Phrases. The NE task requires the system to classify several Named Entities. If there was a more difficult task, an Entity task, which required all Entities to be classified, the system would be more domain independent. It would still be useful to add new lists and grammar rules to switch domains, but the introductory work would have been done. Furthermore, without adding new lists or grammar rules, some output could be generated.

For example, in switching MUC7-Plink from Aircraft Accidents to Launch Events the grammar and the Gazetteer provided no space for rockets. Therefore, rockets could never have arrived as specific semantic output (except when specifically mentioned as a rocket). This is why MUC7-Plink performed so badly on the ST task. It performed better on the TE and TR task because large parts of those tasks (Organizations, Products and People) were accounted for by the grammar and the Gazetteer. If the original system had considered rocket entities, the scores would have been much higher.

There was no Discourse Interpretation work done as part of MUC7-Plink. I simply took advantage of the work done at Sheffield. Clearly, in switching to a new domain, some discourse work would need to be done. However, the amount of work done at Sheffield on the discourse model was also small. To a large degree this work could be considered looking for specific phenomenon in the text, specifically, those phenomena required by the ST, and TR tasks. Perhaps the new SUMMAC tests will provide better insight into a general discourse interpretation mechanism which can easily be culled for specific information, but it seems likely a more sophisticated all-purpose Scenario task would be needed.

References

[1] Advanced Research Projects Agency. 1993. *Proceedings of the Fifth Message Understanding Conference (MUC-5)*, Baltimore, MD. August 1993. San Mateo, CA: Morgan Kaufmann Publishers.

[2] Defense Advance Research Projects Agency. 1992. *Proceedings of the Fourth Message Understanding Conference*, McLean VA. June 1992. San Mateo, CA: Morgan Kaufmann Publishers.

[3] Brill, E. 1994. Some advances in transformation-based part of speech tagging. *Proceedings of AAAI, 1994*

[4] Crain, S. and M. Steedman. 1985 On not being led up the garden path: the use of context by the psychological syntax processor. In Dosty, D., L. Kartunnen and A. Zwicky (eds.) *Natural Language Parsing: Psychological, Computational and Theoretical Perspectives.* New York: Cambridge University Press, pp. 320-358.

[5] Frazier, Lyn. 1983. Processing Sentence Structure. In *Eye Movements in Reading* Keith Ranyor (ed.) New York, NY: Academic Press.

[6] Cunningham, H., Y. Wilks, and R. Gaizauskas. 1996. GATE: A General Architecture for Text Engineering. *CoLing 1996*

[7] Huyck, Christian R. 1994. PLINK: An Intelligent Natural Language Parser. University of Michigan technical report CSE-TR-218-94.

[8] Gaizauskas, R., T. Wakao, K. Humphreys, H. Cunningham, and Y. Wilks. 1995. Description of the LaSIE System as Used for MUC-6. *Proceedings of the Sixth Message Understanding Conference (MUC-6).* San Mateo, CA: Morgan Kaufmann Publishers.

[9] Proctor, P. 1978. Longman's Dictionary of Contemporary English. Longman Group.

[10] Levine, J. R., T. Mason, and D. Brown. 1992 Lex and Yacc. O'Reilly and Associates, Inc.

[11] Lytinen, Steven. 1992 A unification-based, integrated natural language processing system. *Computers and Mathematics with Applications* 23 (6-9), pp. 403-418.

[12] Lytinen, S and N. Tomuro. 1996 Left-corner Parsing for Unification Grammars. *Proceeding of AAAI, 1996*

[13] Marcus, Mitchell P. 1980 *A Theory of Syntactic Recognition for Natural Language* Cambridge, MA: MIT Press.

[14] Pollard, C. and I. Sag. 1994. *Head-Driven Phrase Structure Grammar* Standford, CA:Center for the Study of Language an Information.

[15] Shieber, Stuart M. 1986 *An Introduction to Unification-Based Approaches to Grammar* Stanford, CA:Center for the Study of Language an Information.

[16] Miller, G. 1990. Wordnet: An on-line lexical database *International Journal of Lexicography*, 3(4).

[17] Gaizauskas. R. 1995. XI: A knowledge representation language based on cross-classification and inheritance. Research Memorandum CS-95-24, Dept. of Computer Science, University of Sheffield.