CRL/NMSU and Brandeis: Description of the *MucBruce* System as Used for MUC-4

Jim Cowie, Louise Guthrie, Yorick Wilks
Computing Research Laboratory
New Mexico State University
&
James Pustejovsky, Scott Waterman
Computer Science Department
Brandeis University

INTRODUCTION

Through their involvement in the Tipster project the Computing Research Laboratory at New Mexico State University and the Computer Science Department at Brandeis University are developing a method for identifying articles of interest and extracting and storing specific kinds of information from large volumes of Japanese and English texts. We intend that the method be general and extensible. The techniques involved are not explicitly tied to these two languages nor to a particular subject area. Development for Tipster has been going on since September, 1992.

The system we have used for the MUC-4 tests has only implemented some of the features we plan to include in our final Tipster system. It relies intensively on statistics and on context-free text marking to generate templates. Some more detailed parsing has been added for a limited lexicon, but lack of fuller coverage places an inherent limit on its performance. Most of the information produced in our MUC templates is arrived at by probing the text which surrounds 'significant' words for the template type being generated, in order to find appropriately tagged fillers for the template fields.

OVERVIEW OF THE TEMPLATE FILLING PROCESS

The overall system architecture is shown in Figure 1. Three independent processes operate on an input text. One, the *Text Tagger*, marks a variety of strings with semantic information. The other two, the *Relevant Template Filter* and the *Relevant Paragraph Filter*, perform word frequency analysis to determine whether a text should be allowed to generate templates for particular incident types and which paragraphs are specifically related to each incident type. These predictions are used by the central process in the system, the *Template Constructor*, which uses a variety of heuristics to extract template information from the tagged text. A skeleton template structure is then passed to the final process, the *Template Formatter*, which performs some consistency checking, creates cross references and attempts to expand any names found in the template to the longest form in which they occur in the text. Each of the above processes is described in more detail below.

Relevancy Filters

We have developed a procedure for detecting document types in any language. The system requires training texts for the types of documents to be classified and is developed on a sound statistical basis using probabilistic models of word occurrence [Guthrie and Walker 1991]. This may operate on letter grams of appropriate size or on actual words of the language being targeted and develops optimal detection algorithms from automatically generated "word" lists. The system depends on the availability of appropriate training

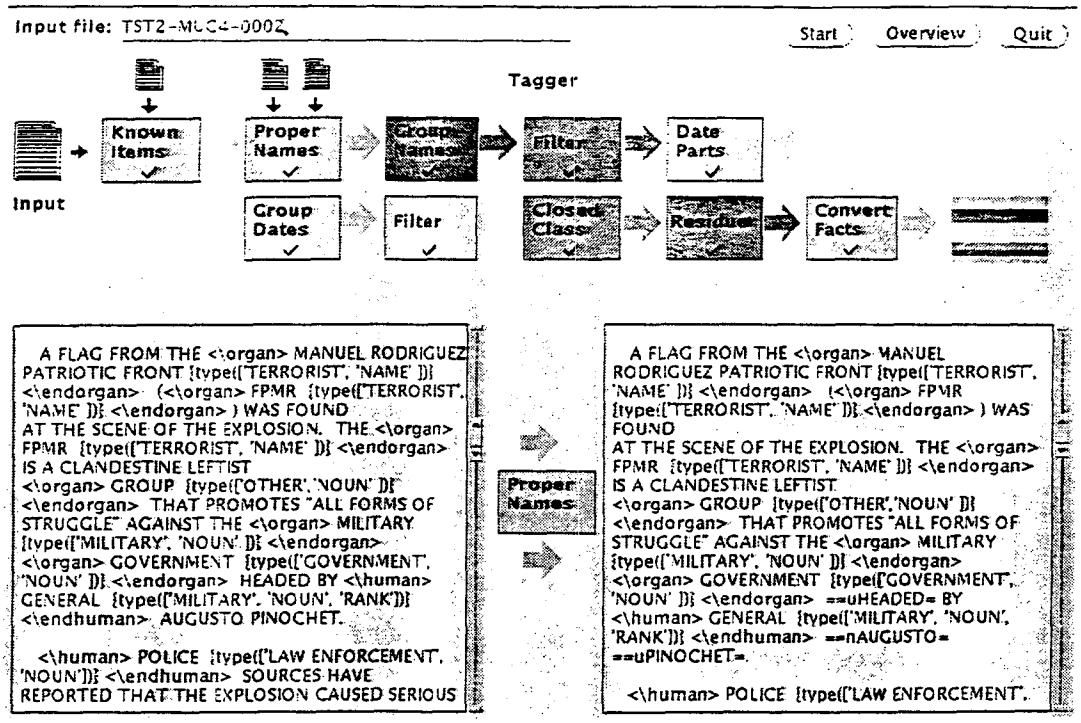

Figure 1: MucBruce - System Overview

texts. So far the method has been applied to English, discriminating between Tipster and MUC texts, and to Japanese between Tipster texts and translations of ACM proceedings. In both cases the classification scheme developed was correct 99% of the time.

The method has now been extended to the identification of relevant paragraphs and relevant template types for the MUC documents. This is a more complex problem due to the non-homogeneous nature of the texts and the difficulty of deriving training sets of text. Each process uses two sets of words, one which occurs with high probability in the texts of interest, and the other which occurs in the 'non-interesting' texts. Due to the complexity of separating relevant from non-relevant information for the MUC texts we actually use three filters, two trained on sets of non-relevant and relevant paragraphs and one trained on sets of relevant and non-relevant texts. The lists of relevant and non-relevant paragraphs were derived using the templates of the 1300 text test corpus. Any paragraph which contributed two or more string fills to a particular template was used as part of the relevant training set; paragraphs contributing only one string fill were regarded as of dubious accuracy and were not placed in either set and all other paragraphs were considered as non-relevant. Word lists were derived automatically by finding those words in the relevant training set which occurred within a threshold of most frequently occurring words in the relevant paragraphs and not in the non-relevant paragraphs, and vice versa to obtain a set of non-relevant words.

The relevant template marker consists of two processes, the first trained on a set of texts consisting of paragraphs from the MUC corpus which produced two or more string fills against text consisting of paragraphs which generated no string fills.

These allow us to determine, based on word counts taken at paragraph level, whether the whole text should be checked for specific template types. The second stage is activated if any single paragraph in the text is found to be 'relevant'. This stage is trained on the set of texts which generated a particular template type against texts which produced no templates. There are separate relevant and non-relevant lists of words used to determine each template type.

The result is a vector represented as a Prolog fact which determines whether the texts will be allowed to generate templates of a particular type. Thus:

FREQUENCY	WORD
135	ELN
128	BOMB
122	KIDNAPPED
77	MURDER
77	MEN
75	MURDERED
75	MORNING
75	MEDELLIN
75	INJURED

Table 1: Part of Relevant Text Word List

FREQUENCY	WORD
136	PEACE
135	I
118	ELECTIONS
104	STATES
96	MARCH
94	UNITED
75	ARENA
73	IF
71	MUST
69	THAN

Table 2: Part of Non-Relevant Text Word List

FREQUENCY	WORD
199	BOMB
115	EXPLOSION
99	INJURED
83	EXPLODED
82	DYNAMITE
65	CAR
65	BOMBS
58	STREET
53	PLACED
49	DAMAGED

Table 3: Part of Relevant Template Word List: BOMBING

FREQUENCY	WORD
174	WERE
112	BOMB
91	ATTACK
72	PEOPLE
69	POLICE
63	SAN
62	DYNAMITE
61	EXPLOSION
60	WHO
54	INJURED

Table 4: Part of Relevant Paragraph Word List: BOMBING

```
slot(4, ['NO', 'ARSON', 'NO', 'ATTACK', 'YES', 'BOMBING',
         'NO', 'KIDNAPPING', 'NO', 'ROBBERY',  'NO', 'DUMMY']).
```

The relevant paragraph filter is the final stage and uses word lists which were derived from relevant and non-relevant paragraphs for each template type.

Once again this operates at the paragraph level and produces a list of paragraph numbers for each template type. These paragraph lists are only used if the relevant template filter has also predicted a template of that type. This stage produces a vector of relevant paragraphs. Thus:

```
rel_paras([[1,3,5],'ARSON', [1,2,3,4,5],'ATTACK', [1,3],'BOMBING',
          [],'KIDNAPPING', [],'ROBBERY', [],'DUMMY']).
```

The two stages can be thought of as first distinguishing relevant texts for a particular template type from among all texts and second, given a relevant text, to distinguish between the relevant and non-relevant paragraphs within that text for the template type.

Partial word lists for relevant and non-relevant texts are given in Tables 1 and 2. The full lists contain 124 and 117 words respectively. Partial relevant word lists for BOMBING at the text level (relevant template) and the paragraph level are given in Tables 3 and 4. The full lists contain 176 and 51 words respectively.

Semantic Tagging

A key question for the Tipster and MUC tasks is the correct identification of place names, company and organization names, and the names of individuals. We now have available to us several sources of geographic, company and personal name information. In addition the templates provided for MUC also supplied name information. These have been incorporated in a set of tagging files which provide lexical information as a pre-processing stage for every text.

The details of the Text Tagger are shown in Figure 2, which is a screen dump of an interface which allows examination of the operation of each stage in the filter. The text window on the left shows the state of a text after the *group dates* process has converted dates to standard form and on the right after the temporary tags placed to identify date constituents have been removed. Each stage, apart from the last, marks the text with tags in the form:

```
<\TYPE> ACTUAL TEXT STRING  {SEMANTIC INFORMATION} <\ENDTYPE>
```

Thus for example a date takes the form:

```
<\date> 5 DAYS AGO {date("14 APR 89",890414)} <\enddate>
```

In general each stage in the pipeline is only allowed to modify text which is not already marked, although an examination of already marked text is allowed. Several stages also place temporary markers in the text

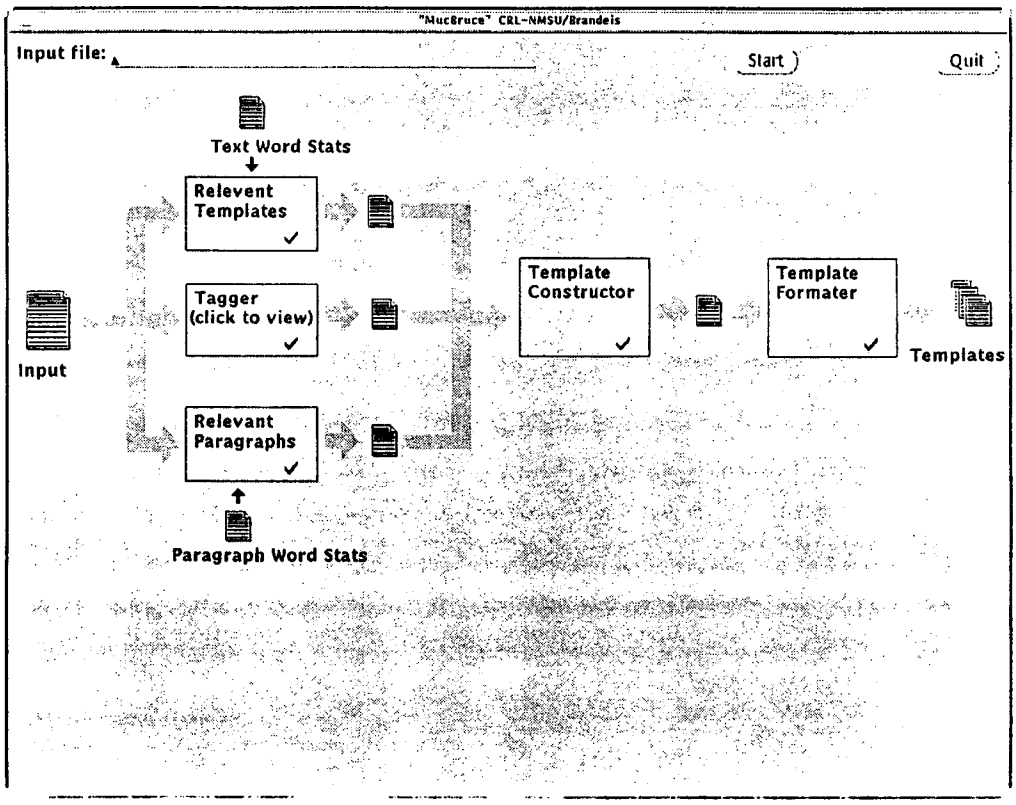

Figure 2: MucBruce - Tagging Pipeline

to allow subsequent grouping by following stages. These temporary markers are removed by the filter stages. Each text is marked as follows:

Known Items Places, organizations, physical targets, human occupations, weapons.

Proper Names Human proper names.

Dates All standard date forms and other references to time.

Closed class Prepositions, determiners and conjunctions.

Residue All other words are marked as unknown.

The final tagged text looks like this:

```
<\name> GARCIA ALVARADO <\endname>, <\num> 56 {num(56)} <\endnum>,
<\cs> WAS {closed(was,[pastv])} <\endcs> <\gls> KILLED
{action(killed,'ATTACK')} <\endgls> <\cs> WHEN
{closed(when,[conj,pron])} <\endcs> <\cs> A {closed(a,[determiner])}
<\endcs> <\weapon> BOMB  {type(['BOMB'])} <\endweapon>  <\res> PLACED
{atom(placed)} <\endres>
<\cs> BY {closed(by,[prep])} <\endcs> <\res> URBAN {atom(urban)} <\endres>
<\organ> GUERRILLAS  {type(['TERRORIST', 'NOUN' ])} <\endorgan>  <\cs>
ON {closed(on,[prep])} <\endcs> <\cs> HIS
{closed(his,[determiner,pron])} <\endcs> <\target> VEHICLE
{type(['TRANSPORT VEHICLE'])} <\endtarget>  <\gls> EXPLODED
```

```
{action(exploded,'BOMBING')} <\endgls> <\cs> AS
{closed(as,[conj,pron,prep])} <\endcs> <\cs> IT {closed(it,[pron])}
<\endcs> <\res> CAME {atom(came)} <\endres>
<\cs> TO {closed(to,[prep])} <\endcs> <\cs> A {closed(a,[determiner])}
<\endcs> <\res> HALT {atom(halt)} <\endres>
<\cs> AT {closed(at,[prep])} <\endcs> <\cs> AN {closed(an,[determiner])} <\endcs>
<\res> INTERSECTION {atom(intersection)} <\endres> <\cs> IN {closed(in,[prep])}
<\endcs> <\res> DOWNTOWN {atom(downtown)} <\endres> <\place> SAN SALVADOR
{type([['CITY','EL SALVADOR'],['DEPARTMENT','EL SALVADOR']])} <\endplace> .
```

For processing by the template constructor the final *convert facts* stage changes each sentence into a Prolog fact, containing sentence and paragraph numbers and a list of structures holding the marked items Thus:.

```
sen(3,3,[name("GARCIA ALVARADO",null),',', num("56",num(56)),',',
cs("WAS",closed(was,[pastv])), gls("KILLED",action(killed,'ATTACK')),
cs("WHEN",closed(when,[conj,pron])), cs("A",closed(a,[determiner])),
weapon("BOMB",type(['BOMB'])), res("PLACED",atom(placed)),
cs("BY",closed(by,[prep])), res("URBAN",atom(urban)),
organ("GUERRILLAS",type(['TERRORIST', 'NOUN' ])), cs("ON",closed(on,[prep])),
cs("HIS",closed(his,[determiner,pron])), target("VEHICLE",type(['TRANSPORT VEHICLE'])),
gls("EXPLODED",action(exploded,'BOMBING')), cs("AS",closed(as,[conj,pron,prep])),
cs("IT",closed(it,[pron])), res("CAME",atom(came)),
cs("TO",closed(to,[prep])), cs("A",closed(a,[determiner])),
res("HALT",atom(halt)), cs("AT",closed(at,[prep])),
cs("AN",closed(an,[determiner])), res("INTERSECTION",atom(intersection)),
cs("IN",closed(in,[prep])), res("DOWNTOWN",atom(downtown)),
place("SAN SALVADOR",type([['CITY','EL SALVADOR'],['DEPARTMENT','EL SALVADOR']])),'.']).
```

All the programs in the Tagger are written in 'C' or Lex. We describe three of these components in more detail.

Known Items

This program uses a large list of known strings which is held alphabetically. For each word in the text a binary search is performed on the list. When a match is found it will be with the longest string beginning with the word, subsequent words in the text are compared with the matched string. If the complete string is matched then this portion of text is marked with the information associated with the string. If a complete match is not achieved the word is checked against the previous item in the list, which may also match the word, and the process is repeated.

The strings and information in the file are derived from a variety of sources. The place name information provided for MUC, organization, target and weapon names derived from the MUC templates and further lists of human occupations and titles derived from Longman's.

Proper Names

The proper name filter uses a variety of methods to successfully identify a large majority of the human names found in a MUC text. It uses two data resources; a complete word list of all the Longman Dictionary headwords and a list of English and Spanish first and last names. In addition it uses the hidden Markov Model algorithm described by BBN in MUC-3 to identify Spanish words. The first stage marks words not in Longman's, Spanish words and known first and last names. The second stage decides whether a group of these items is indeed a name. Any group containing a Spanish word or a known name is recognized, unknown words on their own must be preceded by a title of some kind (identified by the Known Items step). Once an unknown item is identified as a name, however, it is added temporarily to the list of first and last names, so if it occurs in isolation later in the text it will be recognized correctly. A further complication to the problem of name recognition was found in several names which contained text which had already been

identified as a place name. In this case the proper name marker over-rides the previous marking and marks the entire section of text as a human name.

Date Parts

The date marker uses a wide variety of patterns which have been identified in the MUC and Tipster texts as referring to time. Each date is converted to a standard form and the identified text marked. Relative time expressions are always converted with reference to the headline date on the text. This assumption appears to be valid in the vast majority of cases we have examined.

Template Construction

The template constructor uses the tagged text and the list of relevant paragraphs for each template type to generate skeleton templates which are produced as a list of triples, **SLOT NUMBER, SET FILL, STRING FILL**. For example:

```
[[0,'TST2-MUC4-0048',null],
[1,'6',null],
[4,'ATTACK',null],
[2,'19 APR 89',null],
[3,'EL SALVADOR: SAN SALVADOR (CITY)',null],
[6,'null',"BOMB"],
[7,'BOMB',null],
[18,'null',"ROBERTO GARCIA ALVARADO"],
[8,'TERRORIST ACT',null],
[9,'null',"TERRORIST"],
[10,'null',"THE FARABUNDO MARTI NATIONAL LIBERATION FRONT"],
[12,'null',"VEHICLE"],
[13,'TRANSPORT VEHICLE',null],
[19,'null',"GENERAL"],
[20,'MILITARY',null],
[21,'null',null],
[5,'ACCOMPLISHED',null],
[16,'-',null],
[23,'DEATH',null]]
```

A sequence of paragraphs is assumed to generate a new template. The sentences in these paragraphs are examined for a sentence containing a key verb for the template type. Sentences before this sentence are held in reverse order and sentences after in normal order. Each sentence is stripped of any prefatory clause terminated by "that" (e.g. GOVERNMENT OFFICIALS REPORTED TODAY THAT). The remainder of the sentence is reordered into lists containing texts marked with specific semantic types. These correspond to the appropriate fillers for the main sections of the template. The sentence is then marked as active or passive. A search is then made in the current sentence and either the previous or the succeeding ones for items satisfying the appropriate conditions to fill a template slot. Thus for an active sentence the perpetrator will be sought in the head of the sentence and then, if not found, in previous sentences. This provides a crude form of reference resolution as pronouns are not marked with any specific semantic information. The target is checked for in the tail of the sentence and then in subsequent sentences. This process is repeated for all the main fields of the template. It relies heavily on the fact that our text locating techniques are accurate. If no appropriate action word is found the template creation process is abandoned. The process is also abandoned if some of the template filling criteria are not satisfied (eg if the human target is a military officer). The template construction program is written in Prolog and was compiled to run stand-alone using Quintus Prolog.

We obviously need to add more precise syntax and semantics at the sentence level and to provide a structure which allows the inter-relationship of a group of sentences to be captured. The advantage of the

method we are using at the moment is that it is robust and can be used as a fall-back whenever the more precise methods fail. A limited amount of semantic parsing was implemented before the final MUC-4 test. This over-rode the robust method whenever an appropriate parse was found. Due to the limited number of lexical entries we were able to generate before the test, it was not possible to accurately assess the impact of the more precise grammar.

Below are given sample entries of the lexical structures used in the MUC-4 tests. The transitive verb *murder* and gerundive nominal *killing* illustrate the current state of the integration of lexical semantic information (seen in the **qualia** field) with corpus-related information derived from tuning (seen in the **cospec** field) [Pustejovsky 1991]. *Cospecification* is a semantic tagging of what collocational patterns the lexical item may enter into. The **sem** field specifies directly how to map the qualia values into the appropriate slots in the MUC templates.

```
gls("MURDER",
    syn([type(v),subcat1(H1),type(H1,np),subcat2(H2),type(H2,np),
        subcat3(I1),type(I1,np)]),
    qualia([agentive([human(H1)]),formal([human(H2),dead]),
        const([instrument(I1)])]),
    cospec([agentive([np(H1),"*",self]),
        formal([self,"*",np(H2)]),
        const([self,"*","WITH", np(I1)])]),
    sem([type('ATTACK'),perp(H1),hum_tgt(H2),inst(I1),hum_tgt_eff('DEATH')])).


gls("KILLING",
    syn([type(n),subcat1(H1),type(H1,np),subcat2(H2),type(H2,np)]),
    qualia([agentive([human(H1)]),formal([human(H2),dead])]),
    cospec([agentive(["THE",self,"*","BY",np(H1)]),
        formal([self,"OF",np(H2)])]),
    sem([type('ATTACK'),perp(H1),hum_tgt(H2),inst(I1),hum_tgt_eff('DEATH')])).
```

Parsing rules which allow indeterminate gaps are used to match the cospecification against the key sentences found. A parser-generator uses the **cospec** fields of the GLS's to construct the parsing rules, with type constraints obtained from the corresponding **qualia** fields. Certain operators within the rules (such as **np()** and **"*"**) allow varying degrees of unspecified material to be considered in the constituents of the parse. The parsing rules can in this way be seen as specifying complex regular expressions. Because of this looseness, the parser will not break due to unknown items or intervening material.

These parsing rules are individually pre-compiled into compact Prolog code (each a small expression matching machine) before being included into the template constructor. The term-unification machinery of Prolog automatically relates the syntactic constituents of the parse with the type constraints from the **qualia** and also with the arguments of the template semantics, avoiding the need for complex type matching and argument matching procedures.

Performance is degraded by the current partial implementation of the **cospec** field in the lexical structure definition. The statistical-based corpus-tuning program for the lexical structures was not included for the MUC-4 test runs, but is on development-schedule for inclusion in the Tipster test run later this summer. The **cospec** for a lexical item ideally encodes corpus-based usage information for each semantic aspect of the word (e.g. its qualia, event type, and argument structure). This is a statistically-encoded structure of all admissible semantic collocations associated with the lexical item.

The initial seeding of the LS's is being done from lexical entries in the Longman Dictionary of Contemporary English [Proctor et al 1978], largely using tools described in [Wilks et al 1990]. These are then automatically adapted to the format of generative lexical structures. It is these lexical structures which are then statistically tuned against the corpus, following the methods outlined in [Pustejovsky 1992] and [Anick and Pustejovsky 1990]. Semantic features for a lexical item which are missing or only partially specified from dictionary seeding are, where possible, induced from a semantic model of the corpus.

0.	MESSAGE: ID	TST2-MUC4-0048
1.	MESSAGE: TEMPLATE	6
2.	INCIDENT: DATE	19 APR 89
3.	INCIDENT: LOCATION	EL SALVADOR: SAN SALVADOR (CITY)
4.	INCIDENT: TYPE	ATTACK
5.	INCIDENT: STAGE OF EXECUTION	ACCOMPLISHED
6.	INCIDENT: INSTRUMENT ID	"BOMB"
7.	INCIDENT: INSTRUMENT TYPE	BOMB: "BOMB"
8.	PERP: INCIDENT CATEGORY	TERRORIST ACT
9.	PERP: INDIVIDUAL ID	"TERRORIST"
10.	PERP: ORGANIZATION ID	"THE FARABUNDO MARTI NATIONAL LIBERATION FRONT"
11.	PERP: ORGANIZATION CONFIDENCE	*
12.	PHYS TGT: ID	"VEHICLE"
13.	PHYS TGT: TYPE	TRANSPORT VEHICLE: "VEHICLE"
14.	PHYS TGT: NUMBER	*
15.	PHYS TGT: FOREIGN NATION	*
16.	PHYS TGT: EFFECT OF INCIDENT	-: "VEHICLE"
17.	PHYS TGT: TOTAL NUMBER	*
18.	HUM TGT: NAME	"ROBERTO GARCIA ALVARADO"
19.	HUM TGT: DESCRIPTION	"GENERAL": "ROBERTO GARCIA ALVARADO"
20.	HUM TGT: TYPE	MILITARY: "ROBERTO GARCIA ALVARADO"
21.	HUM TGT: NUMBER	*
22.	HUM TGT: FOREIGN NATION	*
23.	HUM TGT: EFFECT OF INCIDENT	DEATH: "ROBERTO GARCIA ALVARADO"
24.	HUM TGT: TOTAL NUMBER	*

Table 5: One of Four Templates Generated for TST2-MUC4-0048

Template Formatting

This final stage is also a Prolog program. This takes as input the lists of triples produced by the previous stage and a list of every name found in the text. It then produces the final template, introducing cross references between serially defined fields which are related to each other. The name list is used to attempt to choose the fullest version of a name found in the text and substitute this for any shorter versions found in the template outline.

TST2-MUC4-0048

MucBruce generates four templates for this text. All are related to the vehicle bomb described at the beginning of the text. The template and relevant paragraphs filters produce the following predictions:

```
slot(4, ['NO', 'ARSON', 'NO', 'ATTACK', 'YES', 'BOMBING', 'NO',
         'KIDNAPPING', 'NO', 'ROBBERY',  'NO', 'DUMMY']).
rel_paras([[1,3,5,6,13,18,19,20],'ARSON',
     [1,2,3,4,5,6,7,8,9,10,11,12,13,14,16,17,18,19,20,21],'ATTACK',
     [1,3,4,5,6,7,8,9,10,11,13,14,16,17,18,19,20],'BOMBING',
     [1,3,6,7,16,17,20],'KIDNAPPING', [19,20],'ROBBERY', [],'DUMMY']).
```

This means that only 4 BOMBING templates will be produced. The first of these produces a reasonably complete match to the key; details on the driver and bodyguards are omitted. The remaining three templates are incorrect, carrying only the information that a bombing has taken place. The attack on the home is not identified by our naive method of multiple template generation, as it already occurs in a sequence of paragraphs in which only the first event is found.

CONCLUSIONS

We feel that our present system, given its only partially completed state, shows potential. In particular the following techniques seem generally useful:

- The recognition of text types and sub-texts within a text using statistical techniques trained on large numbers of sample texts.
- The use of the key templates to derive system lexicons.
- The automatic seeding of lexical structures from machine readable dictionaries.
- The use of lexically-driven cospecification to provide a robust parsing method at the sentence level.
- The successful combination of a variety of techniques in the human name recognizer.
- The production of a number of independent tools for tagging texts.

The system is robust and provides a good starting point for the application of more sophisticated techniques. Given appropriate data it should be possible to produce a similar system for a different domain in a matter of weeks. The tagger software is already being adapted to Japanese and we have already established that we can achieve similar performance with the statistical methods for Japanese texts using character bigrams.

ACKNOWLEDGEMENTS

The system described here has been created using work funded by DARPA under contract number MDA904-91-C-9328. The following colleagues at CRL and Brandeis have contributed time, ideas, programming ability and enthusiasm to the development of the MucBruce system; Federica Busa, Peter Dilworth, Ted Dunning, Eric Eiverson, Steve Helmreich, Wang Jin, Fang Lin, Bill Ogden, Gees Stein, and Takahiro Wakao

BIBLIOGRAPHY

Anick, Peter and Pustejovsky, J. (1990). An Application of Lexical Semantics to Knowledge Acquisition from Corpora. Proceedings of Coling 90, Helsinki, Finland.

Guthrie, Louise and Elbert Walker (1991). Some Comments on Document Classification by Machine. Memorandum in Computer and Cognitive Science, MCCS-92-935, Computing Research Laboratory, New Mexico State University, New Mexico.

Proctor, Paul, Robert F. Ilson, John Ayto, et al. (1978). Longman Dictionary of Contemporary English, Longman Group Limited: Harlow, Essex, England.

Pustejovsky, James (1991) "The Generative Lexicon," *Computational Linguistics*, 17.4, 1991.

Pustejovsky, James (1992) "The Acquisition of Lexical Semantic Knowledge from Large Corpora", in *Proceedings of the DARPA Spoken and Written Language Workshop*, Arden House, New York, February, 1992, Morgan Kaufmann.

Wilks, Y., Fass, D., C-M., Guo, McDonald, J. E., Plate, T. and Slator, B.M. 1990. "Providing Machine Tractable Dictionary Tools," in *Machine Translation*, 5.1, 1990.