GTE: DESCRIPTION OF THE TIA SYSTEM USED FOR MUC-3

Robert Dietz
GTE Government Systems Corporation
100 Ferguson Drive
Mountain View, CA 94039
dietz%gtewd.dnet@gte.com
(415) 966-2825

INTRODUCTION

This paper describes the version of GTE's Text Interpretation Aid (TIA) system used for participation in the Third Message Understanding Conference (MUC-3). Since 1985, GTE has developed and delivered three systems that automatically generate database records from text messages. These are the original TIA, the Alternate Domain Text Interpretation Aid (AD-TIA) and the Long Range Cruise Missile Analysis and Warning System (LAWS) TIA. These systems process messages from the naval intelligence (original TIA) and air intelligence (AD-TIA and LAWS-TIA) domains.

Parallel to the development of these systems, GTE has (since 1984) also been active in Natural Language Processing (NLP) Independent Research and Development. We have developed several systems that build upon the TIA/AD-TIA/LAWS TIA systems. The first system, which processes messages from the naval operations domain, was developed from the original TIA system for participation in the Second Message Understanding Conference (MUCK-II) sponsored by NOSC. The system that this paper describes was developed from the AD-TIA system to overcome weaknesses perceived in the MUCK-II system, as well as in the delivered systems.

GTE's APPROACH TO NLP

The TIA systems are semantics based. Semantic representations are associated with each lexical word/phrase entry in the lexicon, therefore grammatically correct syntactic constructions are not vital for message understanding. This semantic approach allows the system to be output driven. Critical information to be extracted from the text messages drives the semantic data structures constructed by the domain system developer.

TIA's development thrust has been the generation of database records from free formed text. Each message is tokenized, syntactically then semantically analyzed to detect and extract the relevant information needed to construct or update database records. Upon completion of the message analysis, the records (templates) are output in an orderly fashion.

SYSTEM ARCHITECTURE

The TIA used for MUC-3 was developed from the AD-TIA (Alternate Domain TIA). This system, shown in Figure 1, sequentially performs tokenization, syntactic analysis, semantic analysis, and output translation. Each components is discussed below.

Tokenization. The tokenizer finds strings of text delimited by spaces, carriage returns and punctuation marks. It also attempts to classify the string as a known word or a member of a special token class such as a number or a latitude (e.g. `1234N`). The output from the tokenizer is a list of Lisp symbols representing known words and dotted pairs representing a special token class and the text which represents it, e.g. `(:number . 12)`.

If a token is not recognized as either a known word or a member of a special token class, spelling correction is attempted. The spelling corrector is a simple one that looks for transposed, elided or extra letters. If spelling correction fails, the token is classified as belonging to the special token class `:unknown`. For example, in the MUC-3 corpus, the string "Orrantia" is tokenized as `(:unknown . ORRANTIA)`.

Syntactic Analysis. TIA uses a syntactic analysis stage to preprocess tokenized text before attempting semantic analysis. Syntactic analysis finds phrases which may be treated as though they were single words, such as

noun phrases (NPs), and to define synonyms.. As used by the TIA, the syntactic analyzer does not operate at the sentence level of free text, only at the phrase level.[1]

If more than one phrase is possible at a given point in the text, ambiguities are resolved according to the following scheme:

1.) Left to right: the first phrase of overlapping phrases encountered is chosen.

2.) Length: if two phrases start at the same point, the longer of the two is chosen.

3.) Syntactic Priority: A syntactic priority may be assigned to a phrase when it is declared. If two phrases of the same length start at the same point in the text, the one with the higher declared priority is chosen.

The syntactic analyzer retains a parse tree for every phrase that it finds. These parse trees allow the semantic analyzer (see below) to know that, for example, a date was found, but also to extract the month from that date.

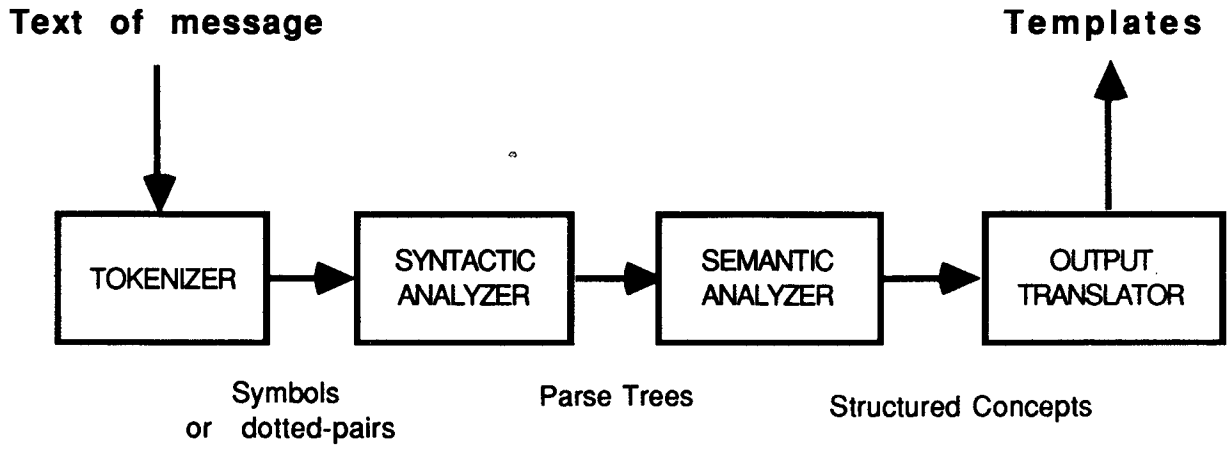

Figure 1: TIA System Architecture

In the MUC-3 domain, the parse trees tended to be rather shallow and consume only a token or two each. For example, the sentences "Police have reported that terrorists tonight bombed the embassies of the PRC and the Soviet Union . The bombs caused damage but no injuries." returns a list of parse trees whose roots are (<LAW-ENFORCEMENT-OR-SECURITY> <REPORT> <DETERMINER> <TERRORIST> <TONIGHT> <BOMBING> <DETERMINER> <EMBASSY> <PREPOSITION> <LOCATION> <CONJUNCTION> <LOCATION> <PERIOD> <DETERMINER> <BOMB> <CAUSE-DAMAGE> <CONJUNCTION> <NO-INJURY> <PERIOD>) .

The syntactic analyzer has the ability to automatically add production rules at run time. For example, the following (simplified) production rules are defined in the system:

```
<location> ::= <district> ?<comma> <location-approx> <location>
<location> ::= <region>
<location-approx>  ::= near
<district>  ::= @<region> district
<region> ::= san isidro
```

[1]The code allows sentential-level parsing, the TIA simply does not utilize the capability.

These rewrite rules allow "Orrantia district, near San Isidro" to be recognized as <location> and cause the following rule to be added to the grammar:

```
<region> ::= orrantia
```

In this example, the question mark indicates an optional item, and the @ sign indicates a place where a new rule might be added. As an example of how phrases are defined in the TIA, <location> above is defined as follows:

```
(def-lexical-entry LOCATION
      :grammar muc3
      :syntax ((<district> ?<comma> <location-approx> <location>)
            <location>
            <region>
            • • •  )
      :predicts ((location-p
              input-text-s (PT-T-STR)
              type-s       (GET-REGION-TYPE
                        (PT-ST-STR '<region>)))))
```

Semantic Analysis. The semantic analyzer (based on concepts embodied in PLUM, developed at the University of Massachusetts by Wendy Lehnert[1]) is the major component of the system. The input to the semantic analyzer is a list of syntactic units identified by the syntactic analyzer. The output of the semantic analyzer is a list of frame-like structured concepts, each consisting of a "head" and an unspecified number of (slot, value) pairs. The semantic analyzer predicts zero or more concepts for each parse tree found by the syntactic analyzer.

Some of the slots in each concept are initialized from information found in the parse tree. For example. <location> predicts location-p, which has slots named input-text-s and type-s.

In the above example, the first location-p predicted has the input-text-s slot filled with "THE PRC" (directly from the parse tree) and its type-s slot filled with 'COUNTRY (by table lookup). Slot initialization information is stored with syntax information, as shown in the sample definition of <location> above.

Other slots are filled by "expecting" information in other frames For example, <bombing> predicts bombing-p, with slots which include agent-s and physical-target-s. Agent-s is expected to be filled by an actor-p, found previously in the same sentence, and physical-target-s is expected to be filled by a theme-p, found later in the same sentence. Passive voice is recognized by the syntactic analyzer, and would have predicted passive-bombing-p, with different expectations about the structure of the text.

Knowledge of expectations is stored in *prediction prototypes*. There is a prediction prototype for each possible type of structured concept.

Concepts are instantiated when "enough" slots are filled. What is "enough" is specific to the individual concept type. Only instantiated concepts can fill slots. Disambiguation is handled by making a prediction for each sense of a phrase and only instantiating the prediction for the correct sense.

That instantiation of a concept can cause actions to occur by allowing calls to Lisp Reference resolution is one example of the type of action that might occur after instantiating a concept.

The concepts are hierarchical.. For example, terrorist-p, which is-a actor-p, can fill the agent-s slot in the bombing-p.

A prediction prototype maintains a list of slots that need to be filled for that type of structured concept, along with information on how to fill them. An example is shown below. This particular concept is predicted by the syntactic constituent <embassy>.

```
(defpred GOVERNMENT-BUILDING-P
 :parents (theme-p)
 :concept-frame (input-text-s
                 country-s = (of-countryp)
                 type-s = "GOVERNMENT OFFICE OR RESIDENCE")
 :control-structure ((expect COUNTRY-S in next location-p
                             input-text-s until PERIOD-P)
                     (expect COUNTRY-S in last countrian-adj-p
                             until PERIOD-P))
 :instantiator (np-instantiator-p))
```

Long term memory consists of lists of concepts, created by semantic analyzer. These lists are called *SYNTAX-MEM*, *CONCEPT-MEM*, *DRAMATIS-PERSONAE*, *STORY-LINE*, and *EVENT-MEM*. The structured concepts in *CONCEPT-MEM* correspond roughly to nouns, those in *EVENT-MEM* to verbs. *SYNTAX-MEM* is a catch-all for miscellaneous structured concepts. *STORY-LINE* is a subset of *EVENT-MEM* in which concepts that refer to the same event are resolved into the same concept . *DRAMATIS-PERSONAE* is a similar subset of *CONCEPT-MEM*. The output of the semantic analyzer is the list of concepts found in *STORY-LINE*.

Output translation. The output translator transforms the internal representations of the concepts extracted from the message into the proper database update template form. This component of the system is tailored to meet the requirements of the particular domain and database under consideration. In the MUC domain, the output translator applies defaults, standardizes fillers for set list slots and performs cross referencing. This module is also responsible for not generating templates. For example, it should determine that a military versus military attack should not generate a template.

The output translator is only partially implemented. It does not incorporate many of the heuristics about generating and combining templates, and incorrectly applies those heuristics that it does incorporate. Additionally, the output translator is dependent on the order in which its input is received. It may generate an entirely different set of template if the list of concepts produced by the semantic analyzer is reversed. This was not intended to be a feature.

REFERENCE RESOLUTION

Reference resolution takes place in two modules: the semantic analyzer and the output translator. In the semantic analyzer, newly instantiated concepts (of specific types) are compared with other similar concepts. If no contradictions are found between a pair of concepts, the two are merged into a single concept. "No contradictions" is defined to mean that 1) both concepts are the same type or the type of one is a direct ancestor of the other in the concept hierarchy and 2) every slot of each matches that of the other, or is empty. Certain slots, such as `input-text-s`, are excluded from the matching requirement, since two concepts may refer to the same entity, but were represented differently in the text. To "match" usually means that the slot fillers are EQUAL, but not always. For example, the `persons-name-s` slot may allow partial matches to succeed; "Smith" matches "John Smith."

The output translator is responsible for combining templates. For example, two attacks at the same time and place should be output as one template. This module is also responsible for deleting multiple templates for a single event. For example, a bombing at a given time and place is an attack or an attack that results in a person being killed is a murder. In each case only one template should be produced. Unfortunately, this module is only partially implemented, so many spurious templates are produced.

CONJUNCTION PROCESSING

Conjunction processing occurs in the semantic analyzer module. The conjunction in the phrase "embassies of the PRC and the Soviet Union." is handled as follows: `<embassy>` predicts `government-building-p`, which when instantiated fills the `physical-target-s` slot of `bombing-p` The concept `government-building-p` has a `slot country-s` which can be filled by a `location-p` concept which has the `type-s` of `'country` and `obj-of-prep-s` of `'of`. The syntactic unit `<preposition>` predicts `preposition-`

p, which upon being instantiated, inserts itself as the obj-of-prep-s slot of the subsequent concept. The first <location>, i.e. "The PRC", predicts a location-p, which meets the constraints needed to fill the country-s slot of the government-building-p. When that slot is filled, the location-p creates a record to indicate that it has filled a slot in the government-building-p. Next, <conjunction> predicts conjunction-p, which, when instantiated, makes a note to try to join the previous concept with the next concept at a later time (the end of the sentence.) The second <location> then predicts a new location-p and the <period> predicts a number of concepts whose only function is to cause other concepts to be instantiated in the proper order. One of these tells the conjunction-p that it's time to try to join the previously noted concepts.

The conjunction joining mechanism verifies that the two locations are the same types of thing. Upon verification, the locations are conjoined by copying concepts in which the first location filled one or more slots, and replacing those fillers with the second location. In this instance, the government-building-p is copied with "the Soviet Union" filling the country-s slot. Since the government-building-p concept filled a slot in bombing-p concept, that bombing-p concept is copied with the new government-building-p filling the physical-target-s slot of the copy.

SUMMARY

GTE believes the TIA architecture described in this paper is sound, robust and practical for a message understanding system. It has in fact been the basis for several delivered systems. It's relatively poor performance in MUC-3 can be attributed to the small amount of time devoted to adapting it to the new domain. The scores were more reflective of the current state of development, especially in the output translation module, than the system architecture.

REFERENCES

[1] Lehnert, W, Narasimhan, Draper, B., Stucky, B. and Sullivan, M., "The Counselor Project - Experiments with PLUM", Department of Computer and Information Science, University of Massachusetts.