SYNCHRONETICS: DESCRIPTION OF THE SYNCHRONETICS SYSTEM USED FOR MUC-3

James Mayfield
Computer Science Dept.
University of Maryland, Baltimore County
Baltimore, MD 21228-5398
mayfield@umbc3.umbc.edu
(301) 455-3099

Edwin Addison
Synchronetics, Inc.
3700 Koppers St., Suite 131
Baltimore MD 21227
76366.1115@compuserve.com
(301) 644-2400

PROJECT BACKGROUND

Synchronetics, Inc., is a startup company in Baltimore founded to develop text processing software products for the commercial and Government sectors. The company, consisting of 7 people, was founded in 1989. Synchronetics had two natural language processing software development projects prior to participation in MUC-3: an off-the-shelf parsing utility called NL-Builder; and a text retrieval system prototype called Text-SR, which was developed under an SBIR contract for Wright Patterson Air Force Base.

Neither of these projects alone was sufficient to handle the MUC-3 problem. Synchronetics was therefore prompted to look elsewhere for additional support. Members that participated on the 'Synchronetics Team' on a volunteer basis[1] were James Mayfield of the University of Maryland, Baltimore County (technical lead and template generation software), Kenneth Litkowski of CL Research of Gaithersburg Md. (software for building the lexicon from a machine-readable dictionary), and Mark Wilson, Roy Cutts, and Bonnie Blades (implementation of the semantic net and phrase and sentence interpretation).

The system was not integrated at the February meeting. At that time static cases were being passed by hand from one processing stage to another. The complete system was fully integrated and running on 100 texts only three weeks before the final submission was due. Because of the relative youth of the system, little time was spent fine-tuning the algorithms and knowledge bases with the 1300 text development corpus. Therefore, we feel that the final results demonstrate the feasibility, but not the potential performance, of our approach.

We estimate that we spent 9 person-months on the development of our MUC-3 system, and that we made use of about 9 person-months of work that was done before we initiated the project. The bulk of the latter time was spent in the development of the NL-Builder product, and in the development of a previous LISP-based version of the KODIAK semantic net representation language.

[1] Synchronetics participation was funded for travel and incidental expenses only—all other labor was voluntary.

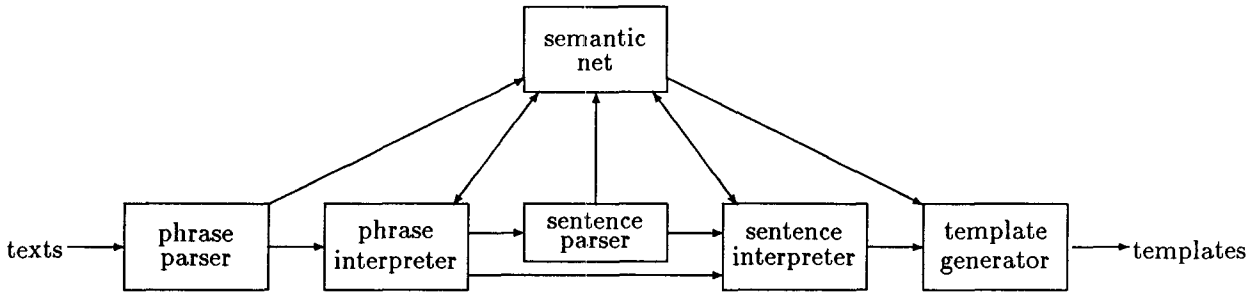

Figure 1: System Architecture

ARCHITECTURE

The Synchronetics system architecture has been strongly influenced by the composition of the Synchronetics team. With team members located at six different sites spread across Maryland, we needed an architecture comprising components that could be developed separately and tested individually.

The Synchronetics system consists of five[2] separate modules that communicate via a semantic net representation language in a pipelined fashion. Each module is a stand-alone program that is written in C and operates on a variety of platforms. Figure 1 depicts this architecture. The five modules are:

1. A phrase parser
2. A phrase interpreter
3. A sentence parser
4. A sentence interpreter
5. A template generator

A semantic net representation language (a variant of the KODIAK language) was developed for use with this project. World knowledge is represented as a single net that is made available to each of the components. In addition, each component passes on to its successor a network description of the text, including all inferences that have been made about the text.

Parsers

It was important to us both to maintain the pipelined architecture (to facilitate the development of different parts of the system at different sites), and to allow feedback from the semantic components of the system to the syntactic components. Therefore, we split the syntactic analysis component into two pieces: a *phrase parser* and a *sentence parser*. The phrase parser is responsible for breaking a text up into words, looking those words up in the dictionary, grouping the words into phrases, and constructing parse trees for those phrases. The sentence parser is a second parser that is responsible for constructing a single parse tree for each sentence in the message. The input to the sentence parser is a sequence of tokens representing the phrases of a sentence as produced by the phrase interpreter. These processes are all performed by the Synchronetics

[2] A number of other components have been implemented or are under development, but were not included in the Phase 2 test.

NL-Builder product.

NL-Builder is a 'programmable' parser. That is, the user may enter and modify the grammar, semantic interpretation rules and morphology, as well as import a dictionary. NL-Builder was used to provide both dictionary tools, and the two parsers. The significant components of NL-Builder are:

- DICTIONARY – The NL-Builder dictionary utilities include morphology rules that are modifiable by the user, a B-tree compiler, and user-specifiable features on the lexical categories.

  Our initial dictionary was an available NL-Builder dictionary with 4000 words in it. It was not matched to the domain, but it contained many common English words. This initial dictionary also included morphological rules, which were left largely unchanged. The dictionary was extended using utilities for dictionary building that are packaged with NL-Builder; these utilities were run on the MUC-3 development corpus. This extension added many domain-specific terms and many slot fill terms and their synonyms. Ken Litkowski then built a system to extract information from the Proximity Linguistic System and enter it into the dictionary by comparing the dictionary with the words in the MUC-3 test corpus. The linking of relevant word senses in the dictionary to the appropriate nodes of the semantic network was done manually.

  The final dictionary consisted of approximately 10,000 word senses and about 30 morphological and tokenization rules. The dictionary was compiled into a b-tree for fast access.

- TOKENIZER – A tokenizer module (which comes as part of the NL-Builder system) is used for marking text into tokens and identifying patterns that may not be in the dictionary (numbers, proper nouns, etc.).

- PARSER – The parser is an extended ATN. It allows a user-specified recursive network state definition with augmented conditions and actions on arcs. In addition, it allows look-ahead tests to prune search paths. Here is an example of a portion of the ATN that handles passive verbs:

```
ARC S_PASSIVE FROM A TO END MATCH VERB
  CONDITIONS
    VERB: FORM.* == VERB:PAST_PARTICIPLE;
    VERB:TYPE.LAST_VERB == VERB:BE;
  ACTIONS
    VOICE = PASSIVE;
    VERB APPEND *;
```

  The parser produces a 'syntactic net' that is stored in the same format as the semantic net. Here is a portion of the syntactic net that is produced by the phrase parser for the sentence (from message 99 of the tst1 corpus):

  > 'Some 3 years ago two Marines died following a Shining Path bombing of a market used by Soviet Marines.'

  Notice that the phrase parser has made a number of errors here, most notably the assumption that 'bombing' is a verb:

```
NP1110063B20
  ISA NP;
  HEAD "SHINING PATH";
  DETERMINER "A";
  NUMBER SINGULAR;
  PERSON THIRD;

VP1110063B40
  ISA VP;
  VERB "BOMBING";
```

Semantic Interpreters

The phrase interpreter is responsible for building a semantic interpretation of each of the phrases discovered by the phrase parser. This process entails mapping from the words in the phrases to the corresponding nodes in the semantic net, then attaching these nodes to each other according to the meaning of the phrase. The sentence interpreter is responsible for building a semantic interpretation of the entire sentence. It uses both the output of the phrase interpreter, and the output of the sentence parser.

Our aim with the semantic interpreters was to make them robust enough to find appropriate connections between the selected nodes in the semantic net even if no explicit semantic interpretation rules are available for the syntactic structure being interpreted. Thus the basis for semantic interpretation is a spreading activation process. If there is a semantic interpretation rule for a given phrase, then that rule is used to connect the nodes in the semantic net representing the components of the phrase. If, however, there is no semantic interpretation rule, spreading activation is used to find plausible connections between concepts.

To continue our example, here is a portion of the phrase interpreter's output for the bombing sentence. Notice that the phrase interpreter has established mappings (via 'SI,' or Semantic Interpretation, links) between the syntactic nodes produced by the phrase parser, and concept nodes in the semantic net:

```
NP1110063B20
  ISA NP;
  HEAD "SHINING PATH";
  DETERMINER "A";
  NUMBER SINGULAR;
  PERSON THIRD;
  SI ORGANIZATION_55;

VP1110063B40
  ISA VP;
  VERB "BOMBING";
  SI BOMB_ACTION_56;

ORGANIZATION_55
  ISA ORGANIZATION;

BOMB_ACTION_56
  ISA BOMB_ACTION;
```

The sentence interpreter must put together an interpretation for the entire sentence. Here is a portion of its output from this sentence:

```
BOMB_ACTION_56
  ISA BOMB_ACTION;
  HAS_PERPETRATOR ORGANIZATION_55 (S:SUBJECT);
```

Notice that the sentence interpreter has identified the Shining Path organization as the perpetrator of the bombing action.

Template Generator

The template generator is responsible for determining which actions that have been represented in the semantic net should lead to the generation of a template, and for the creation of those templates. It begins by examining each potentially reportable action in the semantic net (such as the children of KIDNAP_ACTION, the children of BOMB_ACTION, etc.). For each such action, it tries to determine whether the action falls within the parameters of a reportable action as laid out in the MUC-3 specifications. Since the long-term knowledge stored in the semantic net is currently quite limited, the system usually defaults to reporting the action. Once an action to report has been selected, a template is created for the action, and its slots are filled one at a time. In most cases, slots are filled by starting from the node representing the action being reported, and following a path through the semantic net to another node that stands in the desired relation to the action node. Links are maintained from the syntactic world to the semantic world, so that the system can trace back from a node in the semantic net to the words that caused the creation of that node. For the MUC-3 final test, we attempted to fill only slots 0–7 and slot 11.

Here is the template that is generated for the bombing sentence:

```
0.  MESSAGE ID                      TST2-MUC-3-0099
1.  TEMPLATE ID                     2
2.  DATE OF INCIDENT                25 OCT 89
3.  TYPE OF INCIDENT                BOMBING
4.  CATEGORY OF INCIDENT            TERRORIST ACT
5.  PERPETRATOR: ID OF INDIV(S)     -
6.  PERPETRATOR: ID OF ORG(S)       "SHINING PATH"
7.  PERPETRATOR: CONFIDENCE         REPORTED AS FACT: "SHINING PATH"
8.  PHYSICAL TARGET: ID(S)          -
9.  PHYSICAL TARGET: TOTAL NUM      -
10. PHYSICAL TARGET: TYPE(S)        -
11. HUMAN TARGET: ID(S)             -
12. HUMAN TARGET: TOTAL NUM         -
13. HUMAN TARGET: TYPE(S)           -
14. TARGET: FOREIGN NATION(S)       -
15. INSTRUMENT: TYPE(S)             *
16. LOCATION OF INCIDENT            -
17. EFFECT ON PHYSICAL TARGET(S)    -
18. EFFECT ON HUMAN TARGET(S)       -
```

The date of the incident was not extracted from the sentence, so an incorrect default (the date of the article) was entered. Consequently, the bombing action met the date test, and the template was generated.