USC:
Description of the SNAP System Used for MUC-4

D. Moldovan, S. Cha, M. Chung, K. Hendrickson, J. Kim, and S. Kowalski

Parallel Knowledge Processing Laboratory
University of Southern California
Los Angeles, California 90089-2562
moldovan@gringo.usc.edu
(213)740-4477

INTRODUCTION

Background

The main goal of the SNAP project is to build a massively parallel computer capable of fast and accurate natural language processing [3]. Under NSF funding, a parallel computer was built in the Parallel Knowledge Processing Laboratory at USC and software was developed to operate the machine [2]. The approach in designing SNAP was to find a knowledge representation and a reasoning paradigm useful for natural language processing which exibits massive parallelism. We have selected marker-passing on semantic networks as a way to represent and process linguistic knowledge.

The work for MUC-4 started at the end of January 1992. Prior to this we had implemented on SNAP a simple parsing system accompanied by a small knowledge base. When we started the MUC-4 effort we had only vague ideas how to implement large semantic network knowledge bases for linguistic processing. Thus for us MUC-4 was a major undertaking and soon we realized the difficulties and the time limitation. Faced with the MUC-4 challenge, our group consisting of one faculty and five graduate students spent approximately 1450 hours to engineer a large system.

Approach

The underlying ideas of the SNAP natural language processing system are: (1) memory based parsing, and (2) marker passing on semantic networks [4]. In SNAP, parsing becomes a guided search over the knowledge base which stores linguistic information. Input words activate and predict concepts in the knowledge base. While the semantic network represents the static part of the knowledge base, marker passing is the mechanism which changes the state of the knowledge base with each new word.

SYSTEM ARCHITECTURE

The SNAP architecture is shown in Figure 1.

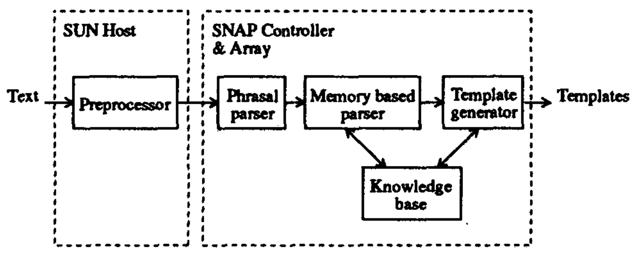

Figure 1: SNAP text understanding system.

input sentence S1:

SALVADORAN PRESIDENT-ELECT ALFREDO CRISTIANI CONDEMNED THE TERRORIST KILLING OF ATTORNEY GENERAL ROBERTO GARCIA ALVARADO AND ACCUSED THE FARABUNDO MARTI NATIONAL LIBERATION FRONT (FMLN) OF THE CRIME.

results of preliminary preprocessing:

SALVADORAN PRESIDENT_ELECT ALFREDO_CRISTIANI CONDEMNED THE TERRORIST KILLING OF ATTORNEY_GENERAL ROBERTO GARCIA ALVARADO AND ACCUSED THE_FARABUNDO_MARTI_NATIONAL_LIBERATION_FRONT (FMLN) OF THE CRIME.

preprocessor output:

```
PRESIDENT_ELECT
1
common-noun PRESIDENT_ELECT sg GOVERNMENT-OFFICIAL
ALFREDO_CRISTIANI
1
proper-noun ALFREDO_CRISTIANI sg HUMAN-NAME
ACCUSED
1
verb ACCUSE pp 1
FMLN
2
proper-noun FMLN sg FMLN
proper-noun FMLN sg ORGANIZATION
```

phrasal parser output:

[SALVADORAN PRESIDENT_ELECT ALFREDO_CRISTIANI]$_{\text{NP}}$ [CONDEMNED]$_{\text{VERB\_SEQ}}$ [THE TERRORIS KILLING]$_{\text{NP}}$ [OF]$_{\text{PREPOSITION}}$
[ATTORNEY_GENERAL ROBERTO GARCIA ALVARADO]$_{\text{NP}}$ [AND]$_{\text{CONJUNCTION}}$ [ACCUSED]$_{\text{VERB\_SEQ}}$
[THE_FARABUNDO_MARTI_NATIONAL_LIBERATION_FRONT]$_{\text{NP}}$ [OF]$_{\text{PREPOSITION}}$ [THE CRIME]$_{\text{NP}}$ [.]$_{\text{PUNCTUATION}}$

Figure 2: Preprocessor and phrasal parser outputs for the first sentence of TST2-MUC4-0048.

PREPROCESSOR

Preprocessor Purpose and Requirements

In the memory based parsing, the knowledge base memory must be used to the greatest possible advantage. For this reason, it was decided to move part of the natural language knowledge base, namely the lexical information, out of the SNAP memory, in order to use SNAP memory for parsing and inferencing. This also necessitates moving the dictionary look-up function for words out of SNAP, and prior to SNAP processing. There were other requirements levied on the preprocessor as well: the preprocessor should recognize previously known noun phrases, a number of contractions and also a number of semi-auxiliary verbs. The preprocessor should detect and block off paragraphs, individual sentence boundaries, and the message headers.

Example of Preprocessing

An intermediate form generated by the preprocessor for S1 of message TST2-0048 is shown in Figure 2. This is before looking up each word in the dictionary. It is not possible to show here, but the preprocessor concatenated all the words of this sentence onto one line. As can be seen, noun phrases like PRESIDENT_ELECT and ALFREDO_CRISTIANI were grouped together into single words. These single words also have corresponding entries in the preprocessor dictionary. By grouping well known noun phrases, including well known human names, a considerable load is removed from the parser. In addition to grouping noun phrases, the preprocessor also groups semi-auxiliary verbs like USED_TO, prepositional phrases like SUCH_AS and AS_TO, and converts contractions like CAN'T and DON'T into CAN NOT and DO NOT. The preprocessor also separates punctuation markers from the words they are attached to. This can be seen in the case of (FMLN), and CRIME at the end of the sentence. Neither FMLN nor CRIME could be looked up in the dictionary unless the punctuation marks were stripped from these words.

The preprocessor failed to group ROBERTO_GARCIA_ALVARADO into a single word, because this name was not known to the preprocessor. This example illustrates the primary failing of the preprocessor. To improve the performance of the preprocessor, the knowledge base that the preprocessor is built from should be enhanced in the domain of interest. It is also necessary to add the corresponding entries, and any other unknown words, to the preprocessor dictionary.

After the above discussed processing, the preprocessor looks up each word in the sentence, and then presents the results to the parser. Examples for four words from the sentence are shown in Figure 2.

For each word, the number of dictionary entries is given, followed by those dictionary entries. Each dictionary entry contains the part of speech, the root of the word, number information for nouns, tense information for verbs, and a semantic definition at the end of the line. These semantic definitions are important, since the parser will mark these nodes in the knowledge-base, and they will be used later in inferencing for template generation.

Preprocessor Implementation and Performance

The preprocessor is implemented solely with the standard Unix commands cat, tr, sed, and awk, using Unix piping with a filtering paradigm. With a dictionary of 16,700 words and 1770 noun phrases, the preprocessor requires approximately 10 seconds to handle a 1.5k byte message. This is not at all bad performance for a largely interpreted process.

KNOWLEDGE BASE

The knowledge base consisting of concept nodes and links between them is distributed over the processor array. The parsing and template filling are performed within the knowledge base by propagating markers through the network.

Linguistic Knowledge Representation

In our memory-based parsing approach, the parsing is performed by matching the linguistic pattern in the input against stored templates in the knowledge base, called *concept sequences.* A concept sequence is a phrasal pattern consisting of a concept sequence root (CSR) and a set of concept sequence elements (CSE) with specific ordering connections.

To provide flexibility, the concept sequences are separated into *basic concept sequences* (BCS) which represent obligatory cases and *auxiliary concept sequences* (ACS) which represent optional cases. Parsing is performed by dynamically combining BCS and ACS. Syntactic information is given by specifying the ordering of CSE, and by attaching a syntactic constraint to each CSE. Semantic information is also given by attaching semantic constraints to each CSE node.

The domain knowledge is represented in a concept hierarchy with is-a relations and various property links between concepts. The concept hierarchy contains concepts which are related to the terrorist domain, such as different event types, target types, and instrument types. It also contains knowledge about different locations in South America and various terrorist organization names. The semantic constraints of the basic concept sequence elements are mapped to one or more of the concepts in the concept hierarchy.

The knowledge base is divided into several *layers.* At the top, there are CSRs which identify the type of interpretation. Below the roots are CSEs which constitute the components of concept sequences. The next levels down are the syntactic patterns and semantic concept hierarchy, which serve as connections between the lexical entries and the CSEs. The lexical layer consists of the input words produced by the preprocessor.

Figure 3 illustrates the layered organization of the knowledge base and how the surface linguistic patterns are mapped to a concept sequence through the syntactic constraints and the semantic concept hierarchy. The **accuse-event** is a CSR, and the **agent, benificiery, object**, and **predicate** are CSEs for the

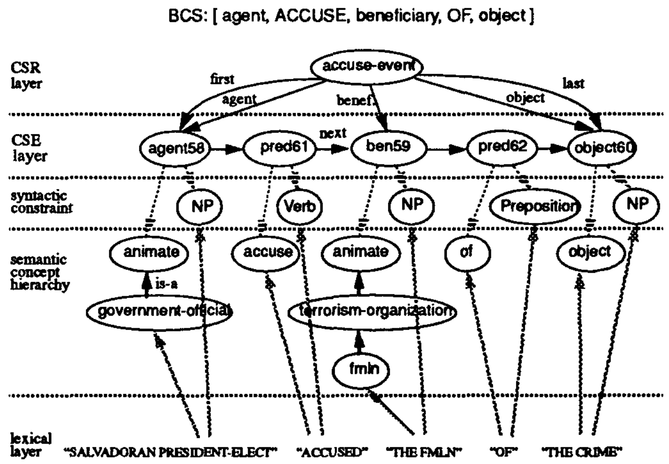

Figure 3: Example of the concept sequence.

accuse-event. Each CSE has its own syntactic and semantic constraints. For example, the syntactic constraint of **agent** is **NP**, and the semantic constraint is **animate**.

The links from the lexical input to those constraints are made in the pre-parsing stage. The SALVADORAN PRESIDENT-ELECT is segmentized as a NP, and its semantic meaning is mapped to **government-official**. When a marker is initiated at the concept **government-official**, it propagates through the concept hierarchy via **is-a** relation to a CSE to activate a concept sequence. Actual memory-based parsing is started by initiating markers from those nodes which are mapped by pre-parsing.

Knowledge Base Components and Size

Currently the entire knowledge base has approximately 12,000 semantic concept nodes and 47,000 links between them. This network of concepts was formed by using a frame-like representation language, converted to a set of SNAP create instructions by a conversion program, and allocated inside the SNAP array by the controller at the loading time. The major components of the knowledge base are the basic concept sequences and the concept hierarchy for event-concepts, domain phrases, and the geographical concepts.

The basic concept sequences occupy 9000 nodes for the CSR, CSE, and the syntactic constraints. The knowledge base contains about 1200 basic concept sequences for about 900 verb entries. Among the 3000 nodes in the concept hierarchy, 1000 nodes are used for the domain concepts including event types, object hierarchy, physical target types, human target types, and instrument types. Another 1000 are used for domain-phrases containing terrorist organization names. The rest are used for geographical concepts containing all the names of countries, cities, and regional names in South America. They are connected vertically to their location types, and horizontally to their location names. For example, San Salvador is connected to **city** as its type, and to **El Salvador** as its location.

PHRASAL PARSER

The phrasal parser takes a sequence of dictionary definitions of words generated by the preprocessor, and groups the words into the following phrasal segments: noun phrase, verb sequence, adverb sequence, adjective sequence, conjunction, preposition, postposition, relativizer, possessive marker ('s), date-time group, and punctuation. The resulting sequence of phrasal segments drives the memory-based parser to get the meaning of the input sentence. The output of the phrasal parser for S1 of message 0048 is shown in Figure 2.

Lexical ambiguity was the major problem in this stage. Most words can have several syntactic categories and semantic meanings. In order to resolve the lexical ambiguity as far as possible, we have derived about 30 heuristic disambiguation rules based on the syntactic information of neighboring words. As a result, most phrasal segments are correctly assigned. However, in case of structural ambiguity or garden path sentences, the phrasal parser yields all possible multiple phrasal definitions, and the next stage, the memory-based parser, performs the disambiguation.

If the lexical entry and its corresponding features are defined in the knowledge base as a lexical layer, and the syntax handling modules are added to the knowledge base, then the input word itself can directly drive the parallel memory-based parser. In this case, a separate sequential phrasal parsing stage can be removed. However, since the maximum number of nodes allowable in the current SNAP computer is 20K, we cannot define all of the lexical entries for the MUC-4 domain in the SNAP knowledge base. Therefore, we had to separate the dictionary from the knowledge base, and the phrasal segmentation based on the dictionary information from the memory-based parsing module.

MEMORY-BASED PARSER

Memory-Based Parsing Algorithm

The memory-based parsing algorithm is based on repeated applications of expectations, activations, and verifications [1]. Two marker types are used for expectations and activations. A *prediction marker* (P-marker) identifies nodes that are expected to occur next. An *activation marker* (A-marker) identifies the node that actually occurred. An expected node in the concept sequence element layer is *verified* only if it receives activation markers through both semantic and syntactic constraint links, whereas an expected node at any other layer is verified whenever it receives an activation marker. Only verified nodes perform an action specific to the node type.

In the beginning of parsing, all concept sequence roots are expected as possible hypotheses, and accordingly their first concept sequence elements are also expected. In effect, we prepare to parse all concept sequences defined in the knowledge base.

When a phrasal input is read, a concept instance is created and a bottom-up activation is propagated from the concept instance to the corresponding subsuming nodes in the knowledge base. Even though many nodes are initially expected, only a small part of them receive activations as input phrases are further processed. Therefore, candidate hypotheses are quickly narrowed down to correct ones. In this way, multiple hypotheses are handled in parallel, and eventually only relevant hypotheses survive as an interpretation of the input sentence.

Parsing Example: processing S1 of TST2-MUC4-0048

Among the basic concept sequences in the knowledge base, [**agent, condemn, object**] and [**agent, accuse, beneficiary, of, object**] are related to S1. We will explain how these two are used to interpret each clause of S1.

As shown in Figure 4, the concept sequence root **condemn-event** and its first concept sequence element **agent1238** are initially expected, which is indicated by P-markers. When the noun phrase [**SALVADORAN PRESIDENT_ELECT ALFREDO_CRISTIANI**] is input to the memory-based parser, a concept instance of the head noun, **alfredo_cristiani#0** is created and connected to the corresponding semantic node **human-name** and syntactic node **NP**. Several links are reserved to represent the relationship between the concept instance of a head noun and instances of noun phrase constituents. In this case, the concept instance of a head noun is connected to the instances of modifiers, **salvadoran#1** and **president_elect#2** via **np_adjective** and **np_noun** links, respectively.

After the concept instance is created, A-markers are moved up from the concept instance **alfredo_cristiani#0** to the subsuming nodes in the knowledge base. Since **human-name** is subsumed not only by **animate**, but also by **object**, two concept sequence elements **agent1238** and **object1239** in the basic concept sequence [**agent, condemn, object**] receive A-markers from both semantic and syntactic constraint nodes. At this stage, only the first concept sequence element **agent1238** is predicted. Therefore, A-marker at **object1239** is neglected, and only **agent1238** is verified and triggers the expectation of the next concept sequence element **predicate1240**. P-marker at **agent1238** and A-markers are now removed, and the new P-marker at **predicate1240** waits for A-markers. The superscripts of P^0, A^1, and P^2 in Figure 4 represent the order of marker activities explained so far.

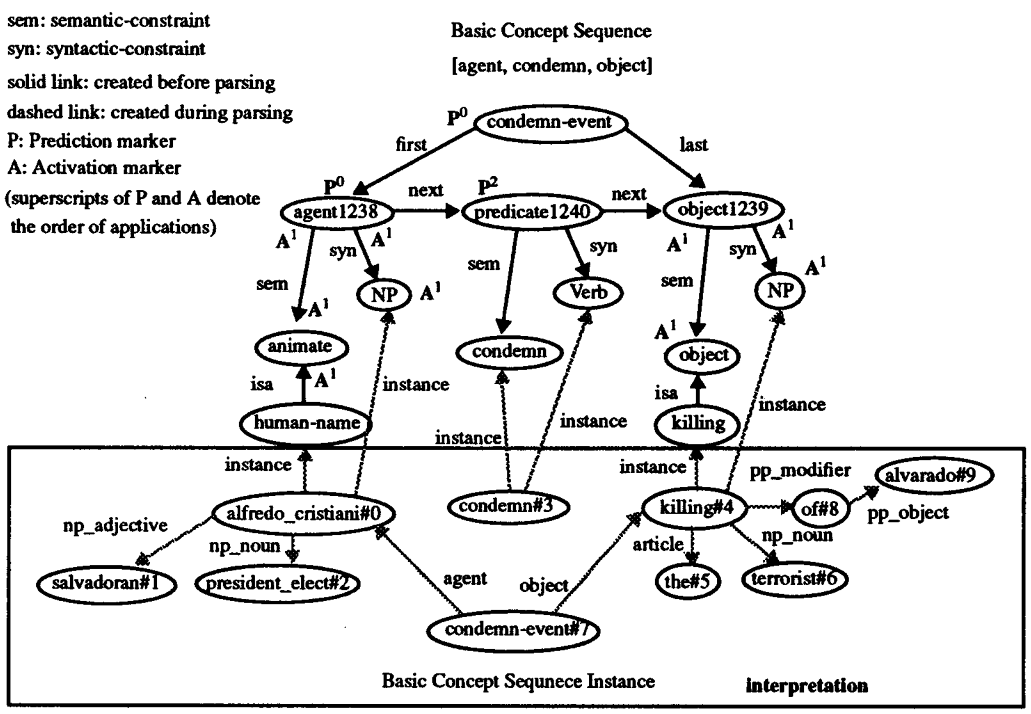

Figure 4: Parsing result of the first clause of S1 is shown with the knowledge base.

The arrival of the next input verb [CONDEMNED] moves activation markers up to the concept sequence element predicate1240, and then triggers the expectation of the next concept sequence element object1239. When the last concept sequence element object1239 receives activation markers from the next input noun phrase [THE TERRORIST KILLING], it sends an activation marker to its concept sequence root condemn-event. Since it was already expected, a basic concept sequence instance condemn-event#7 is created as an interpretation of a clause SALVADORAN PRESIDENT-ELECT ALFREDO CRISTIANI CONDEMNED THE TERRORIST KILLING. This basic concept sequence instance is linked to the corresponding concept instances to represent the meaning of the input sentence. For example, in Figure 4, condemn-event#1 is linked to alfredo_cristiani#0 by an agent link, and to killing#4 by an object link.

The next input phrase is a preposition [OF], which calls the OF-preposition specific rules. As a result, the OF-prepositional phrase is attached to the immediately preceding noun phrase [THE TERRORIST KILLING], which is represented by the pp_modifier link from killing#4 to of#8, and the pp_object link from of#8 to alvarado#9. From these relations, the template generator infers that alvarado#9 is an object of a killing-event by using rules related to the concept killing.

When a conjunction [AND] is input, a conjunction handling routine is called. Since the following phrase is a verb [ACCUSED], AND is recognized as conjoining two verb phrases. As a result, the verb ACCUSED is processed as the beginning of a new clause, and the subject of the preceding verb CONDEMN becomes the subject of ACCUSED. In the knowledge base, this is represented by agent link from accuse-event#19 to alfredo_cristiani#0, as shown in Figure 5.

Parsing Example: processing embedded sentences in S3

Complex sentences are parsed by using the same knowledge base used for simple sentences. When a beginning of an embedded sentence is detected, parsing is guided by different set of markers corresponding to its clause hierarchy, until the end of the embedded sentence. For example, markers P[0] and A[0] are used for the main clause of S3, "GARCIA ALVARADO, 56, WAS KILLED". The conjunction WHEN indicates the beginning of a subordinate clause. Different markers P[1] and A[1] are used for processing the next inputs

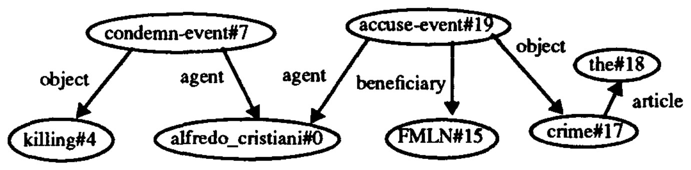

Figure 5: Parsing result of S1.

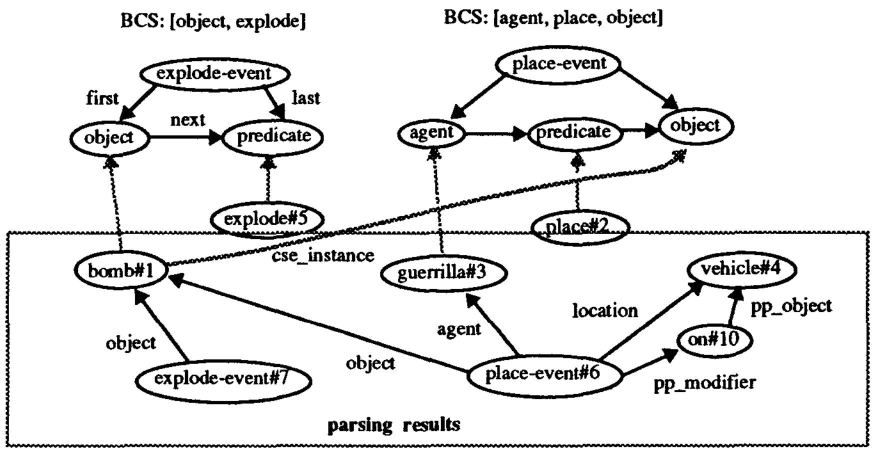

Figure 6: Parsing result of processing "**A BOMB PLACED BY URBAN GUERRILLAS ON HIS VEHICLE EXPLODED**" in S3.

from the phrase [**A BOMB**]. Since the past participle verb "**PLACED**" marks the beginning of an embedded sentence, another set of different markers P[2] and A[2] are used to process the next level of embedded sentence "**PLACED BY URBAN GUERRILLAS ON HIS VEHICLE**", until the verb [**EXPLODED**] indicates returning to markers P[1] and A[1] for the clause of level 1. Therefore, [**EXPLODED**] is processed by P[1] and A[1]. The resulting basic concept sequence instances `explode-event#7` and `place-event#6` are connected to the shared concept instance `bomb#1` by corresponding semantic case links, as shown in Figure 6.

TEMPLATE GENERATOR

Template Generation Algorithm

The memory-based parser generates one or several concept sequence instances (CSI's) for each sentence. These CSI's are collected and processed by the template generator. One template is generated for each newly created CSI. Slots in a template are filled by performing parallel marker propagation, search and set operations directly inside the knowledge base.

The slot filling routine implements some complex rules based on the occurance of "key" verbs and nouns. In the current system, 118 rules (out of 195 rules formulated) have been implemented. The current system considers only a limited set of verbs and nouns. Much more information can be retrieved from the parser's output by using more rules. Some of the rules for a bombing-event are shown below.

RULE 1: If [there is a bombing-event that is not the object of a denial-event, a cease-event, or a stop-event] and [the bombing-event does not have tense future, infinitive, or modal,] then [fill slot 4 (incident type) with BOMBING].

RULE 1.1: If the object of the bombing-event (from RULE 1) is not a <human> or <group-of-humans> then [fill slot 12 (physical target ID) with the object of the bombing-event].

RULE 1.2: If the agent of the bombing-event is an <organization> then [fill slot 10 (perp organization ID) with the agent of the bombing-event].

RULE 1.3: If the agent of the bombing-event is not an <organization> then [fill slot 9 (perp individual ID) with the agent of the bombing-event].

Verbs such as BOMB and MURDER activate the corresponding slot filling routine based on verbs. For example:

S3: GARCIA ALVARADO, 56, WAS KILLED WHEN A BOMB PLACED BY URBAN GUERRILLAS ON HIS VEHICLE EXPLODED AS IT CAME TO A HALT AT AN INTERSECTION IN DOWNTOWN SAN SALVADOR.

The parser produces: BCSI[0] : `kill-event#47, object : 56#45`. The object case of kill-event is human-target name or description and here, GARCIA ALVARADO is human target name. The incident type becomes `attack-event` and the effect of the incident is `death`.

S11: GUERRILLAS ATTACKED MERINO'S HOME IN SAN SALVADOR 5 DAYS AGO WITH EXPLOSIVES.

The parser's result for S11 is BCSI[0] : `attack-event#211, agent : guerrilla#205, object : home#210, time : day#214, location : san_salvador#213`. The object case of `attack-event` is the physical target id, and the agent case is the perpetrator.

After filling slots in each newly generated template, the template is split if necessary into multiple templates. For example if the physical targets are the embassies of the PRC and the Soviet Union, two templates are created.

Control

Newly generated templates are later checked for possible merging. Merging is performed by checking several conditions (incident date, incident location, target's nationality, etc.). For example, in S3, the attack-incident template generated by `kill-event` and the bombing-incident template generated by `explode-event` are merged into one. The attack-incident template of S1 is merged with the bombing-incident template of S3.

Discourse Processing

There is little discourse processing in the current implementation. It is assumed that the discourse is continuing unless the time or location is changed. The message date is kept as a reference time, and the system keeps the discourse time. If there is no explicit change of time, the next sentence is assumed to have the same discourse time. If the time is mentioned explicitly, the discourse time is updated. For example, in S11, 5 DAYS AGO shows the change of discourse time.

There are no inference routines to catch the implicit meaning of sentences. Slots are filled only from the explicit statements using key verbs or nouns. Within a sentence, the parser does not give the relations between events. This is an area of improvement for future versions.

Also, the current implementation does not have reference resolution. Reference resolution is important in discourse processing since it may indicate the connection of texts. For example, in S23 the discourse is returned to the incident of GARCIA ALVARADO who is an attorney general, and this plays a big part in template merging.

S23: ACCORDING TO THE POLICE AND GARCIA ALVARADO'S DRIVER, WHO ESCAPTED UNSCATHED, THE ATTORNEY GENERAL WAS TRAVELING WITH TWO BODYGUARDS. ONE OF THEM WAS INJURED.

REFERENCES

[1] Chung, M. and Moldovan, D., "Memory-Based Parsing with Integrated Syntactic and Semantic Analysis on SNAP", *Technical Report PKPL 91-10*, Dept. of Electrical Engineering-Systems, University of Southern California, 1991.

[2] DeMara, R. and Moldovan, D., "The SNAP-1 Parallel AI Prototype", *Technical Report PKPL 92-3*, Dept. of Electrical Engineering-Systems, University of Southern California, 1992.

[3] Moldovan, D., Lee, W., Lin, C., and Chung, M., "SNAP: Parallel Processing Applied to AI", *IEEE Computer*, May 1992.

[4] Riesbeck, C. and Martin, C., "Direct Memory Access Parsing," *Report 354*, Dept. of Computer Science, Yale University, 1985.