NYU:
Description of the Proteus/PET System as Used for MUC-7 ST

Roman Yangarber and Ralph Grishman
Computer Science Department
New York University
715 Broadway, 7th floor
New York, NY 10003, USA
{roman|grishman}@cs.nyu.edu

INTRODUCTION

Through the history of the MUC's, adapting Information Extraction (IE) systems to a new class of events has continued to be a time-consuming and expensive task. Since MUC-6, the Information Extraction effort at NYU has focused on the problem of portability and customization, especially at the scenario level. To begin to address this problem, we have built a set of tools, which allow the user to adapt the system to new scenarios rapidly by providing *examples* of events in text, and examples of associated database entries to be created. The system automatically uses this information to create general patterns, appropriate for text analysis. The present system operates on two tiers:

- **Proteus** – core extraction engine, an enhanced version of the one employed at MUC-6, [3]
- **PET** – GUI front end, through which the user interacts with Proteus, (as described recently in [5, 6])

It is our hope that the example-based approach will facilitate the customization of IE engines; we are particularly interested, (as are other sites), in providing the non-technical user – such as a domain analyst, unfamiliar with system internals, – with the capability to perform IE effectively in a fixed domain.

In this paper we discuss the system's performance on the MUC-7 Scenario Template task (ST). The topics covered in the following sections are: the Proteus core extraction engine; the example-based PET interface to Proteus; a discussion of how these were used to accommodate the MUC-7 Space Launch scenario task. We conclude with the evaluation of the system's performance and observations regarding possible areas of improvement.

STRUCTURE OF THE PROTEUS IE SYSTEM

Figure 1 shows an overview of our IE system.[1] The system is a pipeline of modules, each drawing on attendant knowledge bases (KBs) to process its input, and passes its output to the next module. The modular design ensures that control is encapsulated in immutable, domain-independent core components, while the domain-specific information resides in the knowledge bases. It is the latter which need to be customized for each new domain and scenario, as discussed in the next section.

The *lexical analysis* module (LexAn) is responsible for splitting the document into sentences, and the sentences into tokens. LexAn draws on a set of on-line dictionaries; these include the general COMLEX syntactic dictionary, and domain-specific lists of words and names. As the result, each token receives a *reading*, or a list of alternative *readings*, in case the token is syntactically ambiguous. A reading consists of

[1] For a detailed description of the system, see [3, 5]

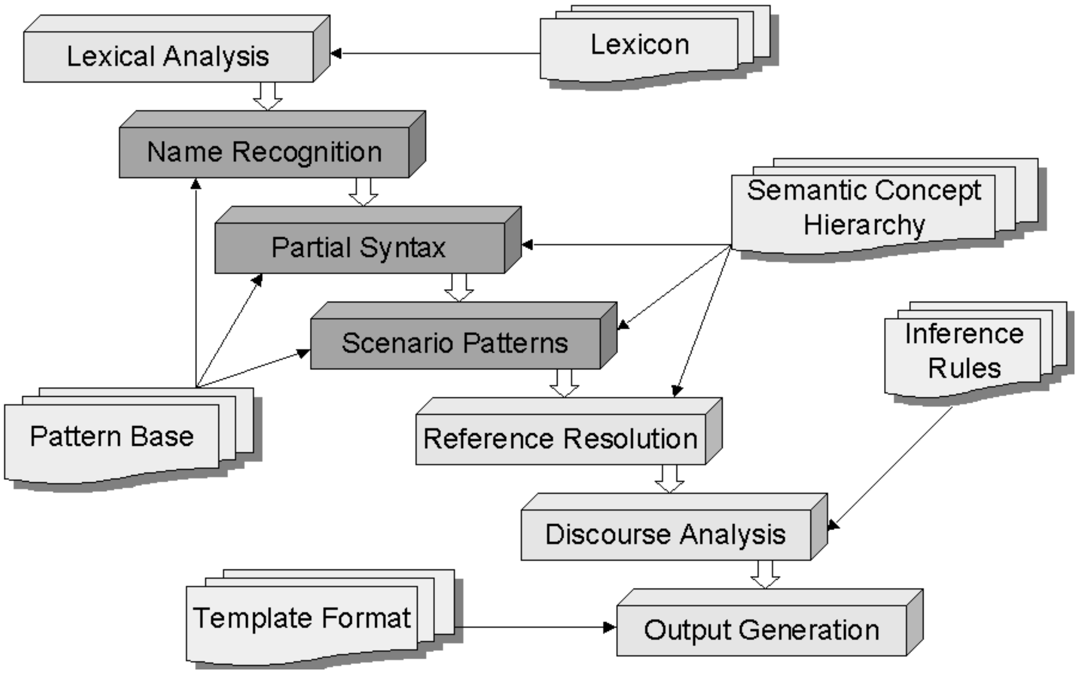

Figure 1: IE system architecture

a list of features and their values (e.g., "syntactic category = *Noun*"). LexAn optionally invokes a statistical part-of-speech tagger, which eliminates unlikely readings for each token.

The next three phases operate by deterministic, bottom-up, partial parsing, or *pattern matching*; the patterns are *regular expressions* which trigger associated *actions*. This style of text analysis, – as contrasted with full syntactic parsing, – has gained the wider popularity due to limitations on the accuracy of full syntactic parsers, and the adequacy of partial, semantically-constrained, parsing for this task [3, 2, 1].

The *name recognition* patterns identify proper names in the text by using local contextual cues, such as capitalization, personal titles ("Mr.", "Esq."), and company suffixes ("Inc.", "Co.").[2] The next module finds small syntactic units, such as basic NPs and VPs. When it identifies a phrase, the system marks the text segment with semantic information, e.g. the semantic class of the head of the phrase.[3] The next phase finds higher-level syntactic constructions using local semantic information: apposition, prepositional phrase attachment, limited conjunctions, and clausal constructions.

The actions operate on the *logical form* representation (LF) of the discourse segments encountered so far. The discourse is thus a sequence of LFs corresponding to the entities, relationships, and events encountered in the analysis. A LF is an object with named slots (see example in figure 2). One slot in each LF, named "*Class*", has distinguished status, and determines the number and type of other slots that the object may contain. E.g., an entity of class "*Company*" has a slot called "*Name*". It also contains a slot "*Location*" which points to another entity, thereby establishing a relation between the location entity and the matrix entity. Events are specific kinds of relations, usually having several operands.

The subsequent phases operate on the logical forms built in the pattern matching phases. *Reference resolution* (RefRes) links anaphoric pronouns to their antecedents and merges other co-referring expressions. The *discourse analysis* module uses higher-level inference rules to build more complex event structures, where

[2] At present, the result of the NYU MENE system, as used in the NE evaluation, does not yet feed into the ST processing.

[3] These marks are pointers to the corresponding entities, which are created and added to the list of logical forms representing the discourse.

Slot	Value
Class	Satellite
Name	–
Manufacturer	Loral Corp.
Owner	Intelsat
...	...

Figure 2: LF for the NP: "a satellite built by Loral Corp. of New York for Intelsat"

the information needed to extract a single complex fact is spread across several clauses. For example, there is a rule that merge a *Mission* entity with a corresponding *Launch* event. At this stage, we also convert all date expressions ("yesterday", "last month", etc.) to starting and ending dates as required for the MUC templates. Another set of rules formats the resultant LF into such a form as is directly translatable, in a one-to-one fashion, into the MUC template structure, the translation performed by the final *template-generation* phase.

PET USER INTERFACE

Our prior MUC experience has shown that building effective patterns for a new domain is a complex and time-consuming part of the customization process; it is highly error-prone, and usually requires detailed knowledge of system internals. With this in view, we have sought a disciplined method of customization of knowledge bases, and the pattern base in particular.

Organization of Patterns

The pattern base is organized in layers, corresponding to different levels of processing. This stratification naturally reflects the range of applicability of the patterns. At the lowest level are the most general patterns; they are applied first, and capture the most basic constructs. These include the proper names, temporal expressions, expressions for numeric entities, and currencies. At the next level are the patterns that perform partial syntactic analysis (noun and verb groups). These are domain-independent patterns, useful in a wide range of tasks. At the next level, are domain-specific patterns, useful across a narrower range of scenarios, but still having considerable generality. These patterns find relationships among entities, such as between persons and organizations. Lastly, at the highest level will be the scenario-specific patterns, such as the clausal patterns that capture events.

Proteus treats the patterns at the different levels differently. The lowest level patterns, having the widest applicability, are built in as a core part of the system. These change little when the system is ported. The mid-range patterns, applicable in certain commonly encountered domains, are provided as *pattern libraries*, which can be plugged in as required by the extraction task. For example, for the domain of "business/economic news", Proteus has a library with patterns that capture:

- entities – organization/company, person, location;
- relations – person/organization, organization/location, parent organization/subsidiary.

Lastly, the system acquires the most specific patterns directly from the user, on a per-scenario basis, through PET, a set of interactive graphical tools. In the process of building the custom pattern base, PET engages the user only at the level of surface representations, hiding the internal operation. The user's input is reduced to

- providing *examples* of events of interest in text, and
- describing the corresponding *output structures* to be created.

$$\overbrace{\boxed{\text{Arianespace Co.}}}^{\textbf{Company}}\;\overbrace{\boxed{\text{has launched}}}^{\textbf{vg}}\;\overbrace{\boxed{\text{an Intelsat communications satellite}}}^{\textbf{satellite}}$$

Figure 3: Initial analysis

Based on this information, PET automatically

- creates the appropriate patterns to extract the user-specified structures from the user-specified text
- suggests *generalizations* for the newly created patterns to broaden coverage.

Pattern Acquisition

The initial pattern base consists of the built-in patterns and the plugged-in pattern libraries corresponding to the domain of interest. These serve as the foundation for example-based acquisition. The development cycle, from the user's perspective, consists of iteratively acquiring patterns to augment the pattern base. The acquisition process entails several steps:

Enter an example: the user enters a sentence containing a salient event, (or copies-pastes text from a document through the corpus browser, a tool provided in the PET suite). We will consider the example *"Arianespace Co. has launched an Intelsat communications satellite."*

Choose an event template: the user selects from a menu of event names. A list of events, with their associated slots, must be given to the system at the outset, as part of the scenario definition. This example will generate an event called "Launch", with slots as in figure 4: *Vehicle*, *Payload*, *Agent*, *Site*, etc.

Apply existing patterns: the system applies the current patterns to the example, to obtain an initial analysis, as in figure 3. In the example shown, the system identified some noun/verb groups and their semantic types. For each element it matches, the system applies minimal generalization, (in the sense that to be any less general, the element would have to match the example text literally). The system then presents the analysis to the user and initiates an interaction with her:

> **np**(C-company) **vg**(Launch) **np**(Satellite)

Tune pattern elements: the user can modify each pattern element in several ways: choose the appropriate level of generalization of its concept class, within the semantic concept hierarchy; force the element to match the corresponding text in the original example *literally*; make the element optional; remove it; etc. In this example, the user should likely generalize "*satellite*" to match *any* phrase designating a payload, and generalize the verb "*launch*" to a class containing its synonyms, (e.g. "fire"):

> **np**(C-company) **vg**(C-Launch) **np**(C-Payload)

Fill event slots: the user specifies how pattern elements are used to fill slots in the event template. Clicking on an element displays its logical form (LF). The user can drag-and-drop the LF, or any sub-component thereof, into a slot in the target event, as in figure 4.

Build pattern: when the user "accepts" it, the system builds a new pattern to match the example, and compiles the associated *action*; the action will fire when the pattern matches, and will fill the slots in the event template as in the example. The pattern is then added to the pattern base, which can be saved for later use.

Syntactic generalization: Actually, the pattern base would acquire much more than the basic pattern that the user accepted. The system applies built-in *meta-rules* [1, 4], to produce a set of *syntactic transformations*

Slot	Value
Class	`Predicate-Launch`
Agent	`entity` $\Longrightarrow$ *<Arianespace>*
Payload	`entity` $\Longrightarrow$ *<Intelsat satellite>*
Site	...
...	...

Figure 4: Event LF corresponding to a clause

from a simple active clause pattern or a bare noun phrase. For this, active example, the pattern base will automatically acquire its variants: the passive, relative, relative passive, reduced relative, etc.[4] Proteus also inserts optional modifiers into the generated variants – such as sentence adjuncts, etc., – to broaden the coverage of the pattern. In consequence, a passive pattern which the system acquires from this simple example will match the event in the walk-through message, "... said Televisa expects *a second Intelsat satellite to be launched by Arianespace from French Guyana later this month* ...", with the help of lower-level patterns for named objects, and locative and temporal sentence adjuncts.[5]

PERFORMANCE ON THE LAUNCH SCENARIO

Scenario Patterns

This section describes how the Proteus/PET system was adapted to accommodate the MUC-7 scenario.

The scenario-specific patterns were primarily of two types: those for launch events ("NASA launched a rocket.", "The terrorists fired a missile.") and those for missions ("the retrieval of a satellite"). Starting from patterns for simple active clauses, the system automatically generated patterns for the syntactic variants, such as the passive, relative, and reduced relative clauses. The missions added information regarding payloads and mission functions to a launch event, but did not directly generate a launch event. In some cases, the mission was syntactically tied to a particular launch event ("... launched the shuttle to deploy a satellite"). If there was no direct connection, the post-processing inference rules attempted to tie the mission to a launch event.

Inference Rules

Consider the event in figure 4: the surface representation contains a generic "*Agent*" role. The agent can be of several types, e.g. it can be a launch vehicle, an organization, or even a launch site, in case the agent is a country. In this case, the role is filled by an organization, which, in principle, further admits the possibility of either the payload owner or the vehicle owner. The scenario specification mandates that the function of the "agent" be precisely specified, although at the surface it is underspecified. In this case, the function can be determined on the basis of the semantic class of the agent, and the observation that the payload-owner slot is already occupied *unambiguously* by another organization entity.

This type of computation is performed by scenario-specific inference rules; in general, this determination can be quite complex. Translating the surface representations into those mandated by the task specification can involve many-to-many relations, such as ones that exist between payloads and launch events, where multiple payloads correspond to a single event, and multiple launch events are concerned with a single payload.

One technique that appeared fruitful in the Launch scenario was extending our set of inference rules with heuristics. Often a slot in an event cannot be filled, as when patterns fail to find a syntactically suitable

[4] The expert user can view the variants which the system generates, and make changes to them directly.

[5] The tools can be used to acquire non-clausal patterns as well, e.g. patterns for noun groups and complex noun phrases, to extend an existing pattern library.

Task	Template Element	Scenario Template
Recall	71	31
Precision	83	68
F-Measure	76.50	42.73

Figure 5: NYU scores on MUC-7 tasks

candidate. The idea was to make intelligent guesses about fillers for these empty slots. For example, consider the first sentence of the walk-through message:

> *Xichang, China, Feb. 15 (Bloomberg) – A Chinese rocket carrying an Intelsat satellite exploded as it was being launched today, delivering a blow ...*

Here we find two similar problems: concerning the launch date and the launch site. Our patterns recognize the corresponding locative and temporal noun phrases, however, because neither stands in a direct syntactic relation to the main launch event clause (here, headed by the verb "explode"), they fail to fill the appropriate slots in the event. We use a simple heuristic rule to recover from this problem: if the launch event has an empty date, and if the sentence contains a *unique* expression of the correct type (i.e. *date*), use the expression to fill the empty slot.

We have experimented with a variety of heuristics for several slots, including organizations for vehicle and payload owners and manufacturers, dates and sites. At present, the contribution of these heuristics to our score accounts for just under 10% of the F-measure. It is also apparent that some of the heuristics actually overgenerate, though we have yet to analyze their effect in detail.

We believe that the overall approach of example-based pattern acquisition is more appropriate than automatic training from annotated corpora, as the amount of training data for ST-level tasks is usually quite limited. We have found pattern editing tool reasonably effective. However, we discovered that much of the task involved creation and tuning of post-processing rules and we do yet not have support in the tool for this activity. This consumed a considerable part of the customization effort . This points to an important problem that needs to be addressed, especially for tasks where the structure of output templates differs substantially from the structure of entities and events as picked up by the syntactic analysis.

We did not specifically focus on the TE task within the launch scenario, and simply used the same system we had used for the ST task. Table 5 is a summary of the scores of our system.

Acknowledgement

We wish to thank Kristofer Franzén of Stockholm University for his assistance during the MUC-7 formal run.

REFERENCES

[1] Douglas Appelt, Jerry Hobbs, John Bear, David Israel, Megumi Kameyama, Andy Kehler, David Martin, Karen Meyers, and Mabry Tyson. SRI International FASTUS system: MUC-6 test results and analysis. In *Proc. Sixth Message Understanding Conf. (MUC-6)*, Columbia, MD, November 1995. Morgan Kaufmann.

[2] Douglas Appelt, Jerry Hobbs, John Bear, David Israel, and Mabry Tyson. FASTUS: A finite-state processor for information extraction from real-world text. In *Proc. 13th Int'l Joint Conf. Artificial Intelligence (IJCAI-93)*, pages 1172–1178, August 1993.

[3] Ralph Grishman. The NYU system for MUC-6, or where's the syntax. In *Proc. Sixth Message Understanding Conf.*, pages 167–176, Columbia, MD, November 1995. Morgan Kaufmann.

[4] Ralph Grishman. The NYU system for MUC-6 or where's the syntax? In *Proc. Sixth Message Understanding Conf. (MUC-6)*, Columbia, MD, November 1995. Morgan Kaufmann.

[5] Ralph Grishman. Information extraction: Techniques and challenges. In Maria Teresa Pazienza, editor, *Information Extraction*. Springer-Verlag, Lecture Notes in Artificial Intelligence, Rome, 1997.

[6] Roman Yangarber and Ralph Grishman. Customization of information extraction systems. In Paola Velardi, editor, *Proc. International Workshop on Lexically Driven Information Extraction*, Frascati, Italy, July 1997. Università di Roma.