Scenario Customization for Information Extraction

by

Roman Yangarber

A dissertation submitted in partial fulfillment

of the requirements for the degree of

Doctor of Philosophy

Department of Computer Science

Graduate School of Arts and Science

New York University

January, 2001

Approved: _____

Professor Ralph Grishman

Dissertation Advisor

To my wife and two cats.

Acknowledgment

I would like to thank my advisor, Prof Ralph Grishman, for his guidance, concern and advice in all matters academic. Without him none of this could have happened.

This research was sponsored in part by the Defense Advanced Research Projects Agency, through the Space and Naval Warfare Systems Center San Diego under grant N66001-00-1-8917. The results and opinions expressed herein do not necessarily reflect the position or policy of the Government, and no official endorsement should be inferred.

I thank the members of my committee, Profs Ernie Davis of New York University and Claire Cardie of Cornell University, for their invaluable time and comments.

I thank Profs Alan Siegel, Richard Cole, and others at the helm of the Computer Science department, who have guided it into being the productive place to work that it is. I hope and trust they will continue to do so for those who come after me.

I thank Pasi Tapanainen and his team at Conexor Oy, in Finland, for the hours that went into making this project work.

I thank my family members, and many friends and colleagues—Rita, Boris, Fania, Suren, Dima, Julia, *et alii*,—for making graduate life almost bearable.

I should thank my step-cat, Misku, and adoptive cat, Ashley, for their thoughtful contributions to this work.

Lastly, I thank Silja for her kindness and support—theoretical, technical, editorial, emotional, and existential.

Abstract

Scenario Customization for Information Extraction

Roman Yangarber

New York University, 2000

Research Advisor: Prof. Ralph Grishman

Information Extraction (IE) is an emerging NLP technology, whose function is to process unstructured, natural language text, to locate specific pieces of information, or *facts*, in the text, and to use these facts to fill a database. IE systems today are commonly based on pattern matching. The core IE engine uses a *cascade* of sets of patterns of increasing linguistic complexity. Each pattern consists of a regular expression and an associated mapping from syntactic to logical form. The pattern sets are customized for each new *topic*, as defined by the set of facts to be extracted.

Construction of a pattern base for a new topic is recognized as a time-consuming and expensive process—a principal roadblock to wider use of IE technology in the large. An effective pattern base must be precise and have wide coverage. This thesis addresses the portability problem in two stages. First, we introduce a set of tools for building patterns manually from *examples*. To adapt the IE system to a new

subject domain quickly, the user chooses a set of example sentences from a training text, and specifies how each example maps to the extracted event—its logical form. The system then applies *meta-rules* to transform the example automatically into a general set of patterns. This effectively shifts the portability bottleneck from building patterns to finding good examples. Second, we propose a novel methodology for discovering good examples *automatically* from a large un-annotated corpus of text. The system is initially seeded with a small set of good patterns given by the user. An incremental learning procedure then identifies new patterns and classes of related terms on successive iterations. We present experimental results, which confirm that the discovered patterns exhibit high quality, as measured in terms of precision and recall.

Contents

List of Figures

List of Tables

List of Appendices

Chapter 1

Introduction

The subject matter of this thesis is *Information Extraction* (or IE), a sub-area of Natural Language Processing (NLP), a sub-area of Artificial Intelligence (AI). Having thus fixed its coordinates on the scientific landscape, we give a 50-word non-technical description of the objective of IE:

1. Process arbitrary free text, such as newspaper or magazine articles, transcribed radio and TV broadcasts or web pages;
2. Find specific types of *facts* reported in the text, e.g., corporate mergers and acquisitions; and
3. Reduce the discovered facts to a structured form, such as a table in a database or a spreadsheet.

Here is an example of a text news segment:

```
Jack Rabbit, Inc. is the hottest new item on the market.
Amid the sudden bidding frenzy for the Pasadena-based
```

```
maker of pancake mix, Dell Corp., the computer giant,
has announced yesterday a bid of US$1.2 billion in its
surprise attempt to acquire the booming startup.
```

This text might be converted by an IE system into a record, as in table 1.1, of "Corporate Acquisitions."

Buyer	*Item*	*Price*	*Date*	*reported-by*	*article-link*	...
Dell	Jack Rabbit	$1.2B	6/2/2000	Reuters	*link to source*	...
...	...	...	...	...	...	...

Table 1.1: Extraction: Sample table of Corporate Acquisitions

Each record in the table can have a link back to the originating document. The exact place in the document where the event is reported may be highlighted for convenient access.

The benefits of transforming unstructured original text into structured form—moving from "textual" to "factual"—are many.

One benefit is ease of manipulation: on plain text the only operation one can perform is keyword search. By contrast, a wide range of operations apply to tables. Given, as above, a table of "All Corporate Purchases" (ever reported in any of one's text sources) the user may issue complex queries, e.g. (as with any table or spreadsheet), to browse the facts in the table, to sort and search the records alphabetically by company, by date of the sale, by the net amount of transaction; compute totals, averages, maxima, etc.

The most important benefit of this conversion is that the tables form a *semantic* index into the text collection. That is an index by meaning, rather than index by

keywords only, which is how most collections have been organized to date. Such a semantic index is a way to impose order on the chaos that is the universe of textual information, and IE technology is a way of imposing such order *automatically*.

IE technology is the result of government-sponsored research over the last 12 years, cf. chapter 2, in its (the government's) initiative to provide automatic tools to news analysts in government agencies who must manually sift through vast collections of textual information for specific facts.

1.1 Central Issues and Concepts

The task of Information Extraction is the focused search for "meaning" in free natural language text[1]. In the context of IE, "meaning" is understood in terms of *facts*, formally described as a fixed set of semantic objects—*entities*, *relationships* among entities, and *events* in which entities participate. The semantic objects belong to a small number of *types*, all having fixed regular structure, within a fixed and closely circumscribed subject domain. This regularity of structure allows the objects to be stored in a relational database.

Another consequence of the regularity of structure in the result of IE is that it makes it possible to establish a notion of a "correct answer". By comparing the system's result against the correct answer, the performance of the system may be easily evaluated. At the very least this evaluation is more direct than in other areas of NLP, such as Machine Translation or Text Summarization. DARPA has conducted several competitive evaluations, called the Message Understanding Conferences (MUCs), cf. chapter 2. The measures of performance used in the MUCs are

[1]The goals of and motivation for IE are discussed in, e.g., [42, 39, 38].

similar to those used in evaluating Information Retrieval, i.e., the score measures recall and precision of extracted database records and fields.

In this thesis, we use the nomenclature accepted in current IE literature; the term *subject domain* denotes a class of textual documents to be processed, e.g., "business news" or "medical reports", and *scenario* denotes the specific topic of interest within the domain, i.e., the set of facts to be extracted. An example of a scenario is "Management Succession," the topic of MUC-6 (the Sixth Message Understanding Conference); in this scenario the system seeks events in which high-level corporate executives left their posts or assumed new ones.

IE systems today are commonly based on pattern matching and partial syntactic analysis.[2] Patterns consist of regular expressions (RE) and their associated mappings from syntactic to logical form. The patterns are stored in a *pattern base*, one of the knowledge bases (KBs) used by the IE system.

Two principal factors are recognized as obstacles in the way of widespread use of Information Extraction today—portability and performance.

- **Portability**: The user needs to be able to apply the IE system to diverse domains and scenarios. Whenever the system is ported to a new scenario, new knowledge bases must be built. Tuning the knowledge bases is recognized to be a time-consuming and expensive process.

- **Performance**: Zipf's Law, a.k.a. the "long tail" syndrome, is a serious problem in IE, as in other areas of NLP. A large number of facts are covered by a small number of frequently occurring patterns, while the remaining facts—the tail of the distribution—are covered by many more rare patterns.

[2] As opposed to full parsing, [17].

To overcome these problems we need to give the IE system developer the ability to quickly customize robust KBs for a new topic, i.e., KBs which have wide coverage and high precision.

1.2 Objectives of the Study

In this work, we focus particular attention on the *pattern base*, as it is recognized to be the most complex by far to customize. We address the customization problem in two stages:

A. First, we introduce a set of tools for building patterns *manually* from examples. To adapt the IE system to a new domain quickly, the user chooses a set of example sentences, or *candidates*, from a training text, and specifies how each candidate maps to a set of semantic objects (a.k.a. logical form). The system then applies *meta-rules* to transform the candidate automatically into a general set of patterns. This effectively shifts the portability bottleneck from building patterns to finding good candidates.

B. Second, we propose a methodology for discovering good candidates *automatically* from a large un-annotated corpus of text. The system is initially seeded with a small set of patterns proposed by the user. An incremental learning procedure identifies new patterns and concept classes on successive iterations.

The overall goal of this thesis is to present an integrated approach to the problem of scenario-level customization. The proposed approach is embodied in an actual IE system, which is used to test it and evaluate its viability.

Chapter 2 contains a brief review of prior work. The following chapter outlines our overall approach to IE. Chapters 4, 5 focus on the details of, respectively, the core IE system and the example-based acquisition tools. Chapters 6–7 present the tools for automatic discovery of patterns. Chapter 8 covers experimental results of discovery, and chapter 9 discusses research problems which remain to be resolved.

Chapter 2

Prior Work

2.1 Information Extraction

For the purposes of our discussion, we shall fix the origin of the time-line at the first so-called Message Understanding Conference (or Competition), MUC-1, held in 1987.

This is not to suggest that there was no work related to IE in the pre-historic times. [24] provides an early indication of the potential importance of extracting relations from text and [47] applies the same ideas to medical texts. However, the field saw a particular surge of activity as the result of the government-funded *MUCs*.

The MUCs, [36, 37, 38, 39, 40], were sponsored by the government, at first by the Navy and later by DARPA, with the explicit objective of stimulating research in Information Extraction. A total of seven MUCs have taken place to date, each MUC focusing on a particular textual domain and topic (scenario), cf. table 2.1.

Each participant built an IE system for the given task. To each participant the

MUC #	*Year*	*Topic*
1	1987	Messages about Naval Operations
2	1989	
3	1991	Terrorism in Latin America
4	1992	(from newspaper and radio broadcasts)
5	1993	Corporate Joint Ventures and Microelectronics
6	1995	Negotiation of Labor Disputes and Corporate Management Succession
7	1998	Airplane Crashes and Rocket/Missile Launches

Table 2.1: MUC History

sponsors distributed the following materials:

1. **Training data**, in the form of source documents and "correct" answer keys, prepared by human annotators.

2. A MUC **scoring tool** which compares the responses produced by the system with the correct answers produced by human annotators.

3. **Test data**—a corpus of documents, which the developers were to keep *hidden*. The test documents were distributed one week in advance of the final submission of results. The sponsors prepared the correct answer keys for the test corpus, but withheld them until after the competition.

The present study builds upon NYU's Information Extraction system called *Proteus*, described in MUC literature, e.g., [21, 22, 17]. In chapter 8, where we

describe our experiments, we will make heavy reference to the two MUC-6 corpora: the 100 documents called the MUC-6 "Formal Training" document set, and the 100-document MUC-6 "Formal Test" corpus. These corpora—the documents together with the corresponding annotations, or answer keys,—were distributed in 1995 in preparation for (Training) and as part of (Test) the MUC-6 competition.

These, and other MUC corpora, are valuable resources, since they were difficult and expensive to produce, requiring domain and linguistic expertise and extensive human labor, for construction and validation. In all of our research, we are keeping the Test corpora strictly hidden, i.e. not revealed to system developers, and used sparingly for benchmarking.

Initially, porting the IE systems to new domains required heavy manual intervention. As the field evolved, emphasis shifted explicitly toward *rapid* development of systems. In MUC-6 and 7, the participants were given four weeks from the time of announcement of topic and release of training data to the time of distribution of test data and formal evaluation. That is a short time, as compared with six to nine months allowed on the earlier evaluations.

Initially, most IE systems relied heavily on parsing. With the clever realization that a complete parse of a sentence may not be at all necessary to find the event described in the sentence, the field saw a general shift toward shallow, partial parsing by means of finite-state pattern matching. In fact, in its quest for a global parse, a full parser may not be able to correctly identify the event and its arguments at all. Parsing ceased to be the prevailing enabling technology, [17, 2].

Over time, the tasks grew more complex. As the understanding of the underlying problems deepened, the overall IE problem was stratified into several sub-problems

or *tasks*. The participants were judged on each task separately, which led to "specialization", where sites chose to compete in some tasks and not others. Today at least five IE tasks are recognized as independent, complex problems:

1. **Named Entity (NE)**: find and categorize (certain classes of) proper names appearing in text.

2. **Template Element (TE)**: find all entities of specified types, (whether named or not), and identify certain immediate features of the entity—e.g., for a person, whether it is a civilian or a military official; for an organization, whether it's a commercial entity or a government agency.

3. **Co-reference (CO)**: find and link together (co-index) all *references* to the "same" entity in a given text.

4. **Template Relation (TR)**: find instances of broader relations among entities, such as the "employment" relation between persons and companies, or the "parent/subsidiary relation" between companies.

5. **Scenario Template (ST)**, the top-level IE task: find instances of *events* or facts of specified types; events are complex relations with multiple arguments, such as a rocket launch, relating the particular rocket, with the date/time/location of launch, with the launching entity, with the cargo carried aboard, with the outcome of the launch—success or failure.

In this thesis, we center almost exclusively on the top-level task, ST. Note that in order to perform well on ST the system must be able to perform all the lower-level tasks. On the other hand, for optimal performance on a higher-level task, optimal

performance on lower-level tasks may *not* be necessary: i.e., to find all events (ST) one need not have to find *all* proper names (NE) in text, just those names that participate in the events that are sought. NE will play an important role in chapter 6; the remaining tasks are outside the scope of this study.

2.2 Other Flavors of IE

In this thesis we focus on extracting facts from what is sometimes termed *unstructured* text. This is contrasted with extraction from *semi-structured* or *structured* text, which contains heavy mark-up, in the form of SGML/HTML/XML/etc. tags, [49].

This is a somewhat different problem from ours, in that it studies different domains and different types of relations from the ones we are interested in. Examples of these are job and apartment advertisements as they appear in classified sections of newspapers. Such passages are typified by a rigid, predictable structure, and commonly contain only one extractable item per passage. By contrast, normal news articles have no formal structural cues, and in general may contain multiple instances of events which we are trying to extract.

2.3 Learning Methods

This section presents a taxonomy of related methods for learning patterns for IE. We observe a trend away from manual construction of patterns and toward automatic and unsupervised techniques.

2.3.1 Semi-automatic Methods

Some prior systems can be classed as semi-automatic, as they require interaction between the user and the system. In [55] we presented PET, some of the interactive tools described in detail in chapter 5 for generalization from examples to extraction patterns. The University of Massachusetts system, AutoSlog, [30, 44] uses a corpus annotated with extraction templates for inducing a "concept dictionary"—the pattern base. AutoSlog requires a human checker to weed out undesirable patterns. Similarly to PET, these approaches exhibit a person-in-the-loop flavor. We should also place in this category [10], which proposes a comprehensive environment for building IE systems, by integrating iterative user filtering of results from an automatic pattern learning algorithm. PET constitutes somewhat of an advance over [30], in that it provides a richer interface, in particular one which allows the user to control a concept hierarchy and use it for generalization, as well as providing for automatic syntactic generalization.

2.3.2 Automatic Methods

[13, 34] have focused on methods for automatically converting a corpus annotated with extraction examples into extraction patterns. [34, 14] report on experiments with hidden Markov models. RAPIER, in [8, 9], builds symbolic rules for identifying slot fillers using Inductive Logic Programming (ILP). The shortcoming of these approaches is that they do not reduce the burden of *finding* the examples to annotate. They shift the portability bottleneck from the problem of building patterns to that of finding good candidates, while the system needs relatively large amounts of annotated training data, even for relatively simple extraction tasks. For example,

the automated training of the University of Massachusetts system [13] performed well when trained on the vast MUC-4 corpus (1500 articles), but suffered a substantial drop in performance on MUC-6 training (100 articles). One reason for the need for large training corpora is that most such systems learn template-filling rules stated in terms of individual lexical items, although recent work from NTT [48] describes a system capable of specializing and generalizing rules stated in terms of a pre-existing hierarchy of word classes.

2.3.3 Active Learning

Automatic learning from an annotated corpus can quickly pick up the most common patterns, but requires a large corpus to achieve good coverage of the less frequent patterns. Because of the typically skewed distribution of patterns in a corpus—with a few patterns appearing frequently, followed by a long tail of rare patterns,—the user who undertakes *sequential* annotation of examples for automatic learning finds him/herself annotating the same examples over and over again. *Active learning* methods try to cut down the number of examples the user must annotate by selecting suitable candidates. [49, 53] report some gains in learning efficiency by selecting for annotation those examples which match patterns which are similar to good patterns, or examples which match patterns about which there is considerable uncertainty (patterns supported by few examples).

2.3.4 Learning from Un-annotated Corpora.

The DIPRE system [7], and the Snowball system [1], modeled after the DIPRE algorithm, use bootstrapping to find patterns starting with only a small set of

examples and with no pre-annotated data. The process is seeded with a set of tuples in some given (binary) relation, e.g., *book/author* and *organization/location-of-headquarters*. The algorithm then searches a large corpus for patterns in which one of these tuples appears. Given these patterns, it can then find additional examples which are added to the seed, and the process is then repeated. This approach takes advantage of facts or events which are stated repeatedly and in different forms in a large corpus, such as the web. One limitation of the reported approach is that the patterns use only surface information. A more inherent limitation is that the scheme for ranking tuple relevance relies on the fact that the relation is functional.

These publications in particular, and others focusing on learning from unannotated corpora constitute the prior work which most closely relates to our study. [11] iteratively trains two weak, mutually redundant and independently sufficient classifiers for name classification. [5, 35] bootstrap from unclassified web pages to find home pages of university professors and courses. [51] and [59] are among the earlier pioneers of the bootstrapping approach; this work is of particular simplicity and lucidity. [51] shows how bootstrapping can effectively train a "concept spotter", a classifier for proper names or noun phrases, by learning patterns stated in terms of short-range lexical items. [59] uses bootstrapping for word-sense disambiguation; though it's a somewhat different problem from ours, the ideas have important similarities—which attest to the generality and strength of the overall principles that unify all this research. [45] proposes automatic methods for filling slots in event templates, their task being closest to ours; a related publication, [46] finds word classes relevant to a scenario and patterns for finding these word classes.

In [45] all possible patterns (a word and its immediate syntactic context) are

generated for a collection containing both relevant and irrelevant documents. Patterns are then ranked based on their frequency in the relevant vs. the non-relevant parts of the collection, preferring patterns which occur more often in the relevant documents. The top-ranked patterns turn out to be effective extraction patterns for the task. In that respect this work is quite closely related to ours (our formula for computation of patterns scores is based on the formula presented in this paper).

However, Riloff's work differs from ours in several crucial respects.

1. First, while AutoSlog-TS is built on the observation that "domain-specific expressions will appear substantially more often in relevant texts than irrelevant texts", it does not exploit the *dual* relationship between relevant documents and good patterns. This forces its reliance on a pre-classified corpus, a constraint which is removed in our work.

2. Riloff's systems do not attempt to recover entire events, after the fashion of MUC's highest-level scenario-template task. Rather the patterns produced by these systems identify NPs that fill *individual* slots, without specifying how these slots may be combined at a later stage into complete event templates. Our work focuses on directly discovering *event-level*, multi-slot relational patterns.

3. Riloff's approach either relies on a set of documents with relevance judgements, [45], or utilizes an un-classified corpus containing a very high proportion of relevant documents, [46]. By contrast, our procedure requires no relevance judgements, and works on the assumption that the corpus is balanced and the proportion of relevant documents is small.

Classifying documents by hand, although admittedly easier than tagging *event* instances in text for automatic training, is still a formidable task. When we prepared the test corpus, it took 5 hours to mark 150 short documents.

Chapter 3

Overview of the IE System

Our IE system can be logically divided into three phases:

I. **Proteus**: a back-end core IE engine. The core engine operates by regular expression (RE) pattern matching. It draws on attendant knowledge bases (KBs) of varying degrees of domain-specificity.

II. **PET**: a GUI front end to the core IE engine. PET enables the user to modify the knowledge bases to tune the system to a new domain and scenario.

III. **ExDisco**: a suite of auxiliary tools, to aid the user in the otherwise highly laborious process of tuning the knowledge bases. We are focusing on automatic techniques for creating the knowledge bases, without manually constructed training corpora.

Phase I is an existing, self-contained lower-level component, implemented by Prof. Ralph Grishman at NYU for participation in MUC competitions, details in [19, 17]. Phases II and III provide higher-level functionality to the overall IE

system, and form the substance of this thesis. We developed Phase II to address the problem of customization. Our experience with the interactive tools exposed the need for *automatic* corpus analysis, to provide a comprehensive approach.

These components are listed chronologically, in the order of their emergence. Early IE researchers wrote Lisp or Prolog or Perl code to build monolithic extraction systems. Then the KBs were factored out to make the systems portable to new scenarios, and convenient graphical editors were developed for each KB. Then the question arose: how does one populate the KBs—short of the inevitable *manual* search through vast corpora for typical relevant passages, at best aided by standard IR techniques to narrow down the search space?

This is the question this research aims to answer.

3.1 Proteus: the Core IE Engine

The system consists of a cascade of modules with their attendant knowledge bases. Each module transforms the input text document. For a detailed discussion of the system, see [17, 19].

There are four customizable knowledge bases in Proteus:

- Lexicon: contains scenario-specific terms
- Concept base: groups terms in a hierarchy of classes
- Predicate base: describes the logical structure of events to be extracted
- Pattern base: contains RE patterns that fire on events in text

Generally, the bases contain a domain-independent and a domain-specific component. We have undertaken extensive research that seeks disciplined methodology for customizing the domain- and scenario-specific components of the knowledge bases.

Of these knowledge bases, by far the most complex is the pattern base. The patterns perform the most complex IE function: mapping from the syntactic domain to the semantic. Therefore, most of the following discussion focuses on the problems associated with discovering, building and generalizing patterns. We deem this the top-level task—other knowledge bases and problems associated with customizing them are considered insofar as they interrelate with the problem of pattern customization.

3.2 PET: Interactive Customization

Given a functioning core IE engine, the problem of customization becomes central: the user needs the ability to adapt the system to diverse scenarios. Customizing the knowledge bases manually is time-consuming and requires considerable IE expertise and familiarity with the workings of the system. To alleviate the expertise bottleneck, we have implemented Proteus Phase II, described in [55, 56], a suite of knowledge-base editors and document/annotation browsers.

PET provides tools for the user to build patterns *from examples*. To customize the pattern base to a new domain quickly, the user chooses a set of example sentences, *candidates*, in a training text, and for each candidate specifies the logical form that it induces. The system then applies *meta-rules* to transform the candidate automatically into a general set of patterns.

PET allows the *user* (i.e. scenario developer) to start with candidates in text which contain events of interest, the candidates, and generalize them into patterns. However, the user is ultimately responsible for *finding* all the candidates, which usually amounts to searching through example sentences in a very large training corpus. This search is performed manually, at best aided by keyword-based queries to narrow down the search space. Should the user fail to provide an example of a particular class of syntactic/semantic construction, the system has no hope of recovering the corresponding events. Our experience has shown that

(1) the process of discovering candidates is highly expensive, and

(2) gaps in patterns directly translate into gaps in coverage.

In effect, PET thus shifts the portability bottleneck from building patterns to finding examples. How can the system help automate the process of *discovering* good examples?

3.3 ExDisco: Automatic Discovery of Patterns

Our experiments with automatic pattern discovery are presented in [57, 58, 23]. The system aims to find examples of all common linguistic constructs relevant to a scenario. This is not a simple matter of counting, however, since we are (typically) not provided in advance with a sub-corpus of relevant passages—these must be found as part of the discovery process. The difficulty is that one of the best indications of the relevance of the passages is precisely the presence of these constructs. Because of this circularity, we introduce a procedure that acquires the constructs and passages in tandem.

It is important to observe that the pattern base and the concept base *together* serve to define the set of constructs (phrases or clauses) that the system will extract. We are seeking paraphrases for a certain type of event, and these paraphrases can occur at the level of single words as well as at the level of complex syntactic constructions, such as clauses. Also, there is a certain tradeoff between patterns and concepts. For example, if two patterns differ by a single constituent, they can be unified simply by introducing a new concept class that unifies the classes of the two divergent constituents. Thus an effective pattern discovery procedure must take into account concept variation.

In the following three chapters we describe each of these phases in turn: Proteus, PET, and ExDisco.

Chapter 4

Proteus: Core System

4.1 Architecture

A diagram of our IE system [17, 55, 56] is shown in figure 4.1. Logically, the system is a pipeline of modules: each module draws on attendant KBs, processes its input, and passes output to the next module.

Physically, the modules operate on the internally stored document text by adding *annotations* to it, [18]. Annotations are structures attached to spans of tokens—segments of text—which contain syntactic and semantic information relating to these spans, for use in the subsequent phases of processing.

Each module can be viewed as solving a (potentially independent) sub-problem, in the overall quest for meaning that is IE, and indeed there is a considerable body of research dedicated to each individual sub-problem. The problems become harder as we move down the pipeline, and each stage has the potential to contribute (and in practice does contribute) errors to the overall final result. Dealing with the problems which each of the modules encounters is beyond the scope of this thesis. Here we

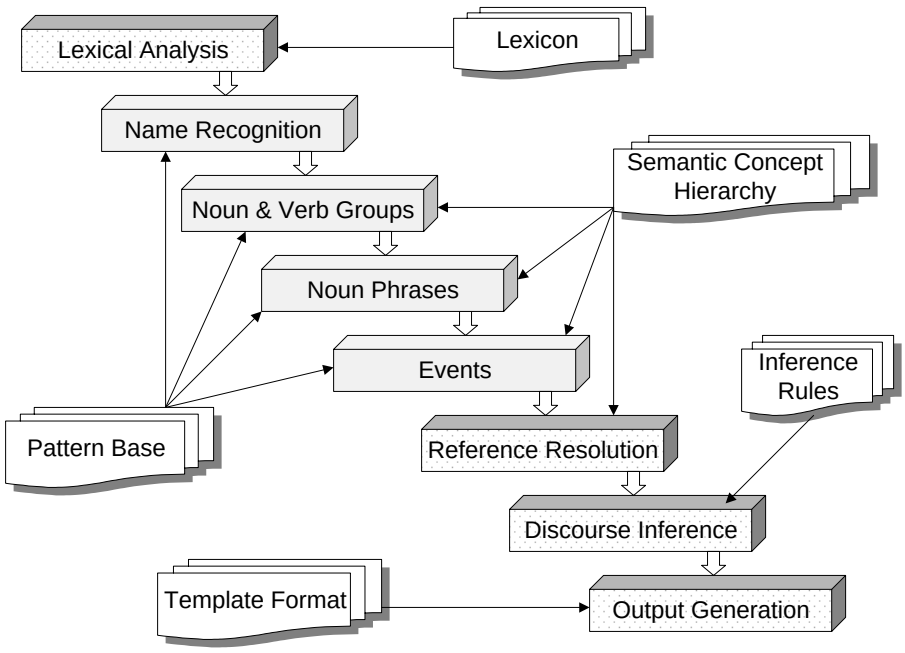

Figure 4.1: Proteus system architecture

focus almost exclusively on the problem of building and finding high-level scenario patterns; thus we implicitly consider all other problems as "solved"—though in reality they are far from it.

4.1.1 Lexical Analysis

The *lexical analyzer* is responsible for

- breaking a document into sentences,
- breaking sentences into tokens,
- looking up the tokens in dictionaries, and
- (optionally) assigning syntactic categories, such as part-of-speech tags, to the

tokens.

This module draws on a set of on-line dictionaries and heuristic rules.

An important factor in IE is what type of text we are processing. We observe that text types fall on a continuum along the dimension which corresponds to, for lack of a better term, its "cosmetic" layout; the text may:

- be heavily marked up with structural information by SGML or XML, e.g., explicit paragraph and sentence boundaries, name types, etc. This is typically the case with Web pages, or electronic newspaper articles;
- be plain, with minimal or no mark up, but contain correct, dependable punctuation and capitalization. This can be, e.g., printed matter digitized by OCR software;
- have minimal or no punctuation and case differentiation. This includes text transcribed by speech recognition software, such as radio or TV news broadcasts, or free dialog.

We can use the term "cosmetic" because to a human reader the presence or the absence of mark-up has marginal effect on the text's understandability. Human language has sufficient redundancy built into it to enable the human reader to extract meaning from it even in the absence of punctuation, prosody, or other extra-lexical means. From a computational point of view, though, the cosmetic can make all the difference.

Texts used in the MUCs fall into the first category. A sample SGML document is shown in appendix A. This document contains a (paraphrased) text segment from the MUC-6 development corpus, which was drawn from a collection of Wall

Street Journal articles. Clearly, the less "cosmetic" information is available in the text, the harder the job of the IE engine, and in particular of the lexical analyzer.

The sentence breaker is a non-trivial problem, as, in general, one cannot simply assume that the sentence ends at a period or an exclamation point. Even if capitalization is present, there are acronyms and abbreviations which can easily confuse a naive algorithm. On the other hand, sentence identification may not be strictly necessary, since, as in the case of Proteus, the system does not attempt to construct a global parse of a sentence, and in principle the entire document could be processed at once.

Lexical analysis attaches to each token a *reading*, or a list of alternative *readings*, in case the token is syntactically ambiguous. A reading contains a list of features and their values (e.g., "syntactic category = *Noun*"). This module also attempts to find some idiomatic expressions.

In general, deciding on the appropriate reading for a token is not a simple matter of looking up in a list. The Proteus lexical analyzer incorporates a statistical part-of-speech tagger, which ranks readings for each token by statistical likelihood.

4.1.2 Name Recognition

The *name recognition* module is responsible for identifying proper names in the text, and attaching semantic categories to them. Proper names can fall into many categories; many objects in discourse can be named:

- persons; e.g.,
  - "George Washington", "George", "Washington", "Calvin Klein",

- geographic locations or geo-political entities:
  - "Washington, D.C.", "Washington State", "Washington"
- companies and organizations:
  - "IBM", "Sony, Inc.", "Calvin Klein & Co.", "Calvin Klein"
- products and artifacts:
  - "DC-10", "SCUD", "Barbie", "Barney" (the Purple Dinosaur)
- works of art:
  - "War and Peace", "Mona Lisa", "Gone with the Wind"
- other groups, such as sports teams, musical bands, orchestras, etc.:
  - "the Boston Philharmonic", "the Boston RedSox", "Boston", "Washington State"
- major events, political, economic, meteorological, etc:
  - "Korean War", "Million Man March", "Cuban Missile Crisis", "The Great Depression", "The Dark Ages", "The Ice Age", "El Niño", "Hurricane George"
- laws, regulations, legal cases:
  - "the Equal Opportunity Act", "Row v. Wade"

For some categories, simple local contextual cues may available, such as:

- capitalization;

- special markers, such as personal titles ("Mrs.", "Esq."), and company designators ("Inc.", "Ltd.");
- specialized *closed* lists, e.g., a list of common personal first names, a gazetteer, a list of company names;
- patterns which encode heuristic rules; e.g., the rule

  `Location-Name` $\Longleftarrow$ `Direction-Phrase + Unknown-Name`

  might be used to categorize the phrase "Western Atlantis" as a location, without prior knowledge of a place called "Atlantis."

However, the general problem is quite complex, and one must rely on longer-range surface cues or deeper semantic information to find and categorize names. One problem is resolution of multiple ambiguities, such as those appearing in the examples above. A related problem is *aliasing*, since typically a name will appear throughout a document multiple times and in variant forms, with shorter versions or acronyms replacing the full form.

Name recognition is a heavily-researched topic, with the some systems today reaching 96% accuracy in *narrow* domains. For example, [6] describes a statistical method for building a NE recognizer automatically. It uses Maximum Entropy estimation to compute probabilities of boundaries and classifications of named entities, though it requires training data pre-tagged by human annotators. The NE recognizer we use in Proteus is rule-based, and we will revisit this component in greater detail in later sections which deal with automatic pattern discovery.

4.1.3 Higher-Level Patterns

The next module uses primarily *syntactic* patterns to identify small syntactic units, such as basic noun groups (NG), which are nouns with their left modifiers, and verb chains or verb groups (VG), which consist of a head verb preceded by modals or adverbials. When a pattern identifies a NG or VG, the system marks the group with syntactic information, such identifying the head word of the group. In case of an NG, Proteus also marks it as a basic noun phrase (NP), creates the corresponding semantic structure, its *entity* (see below), and links the NP to the entity.

The next phase—incrementally building on information gathered at the earlier phases—marks larger, higher-order NPs. We distinguish basic NGs from NPs which can contain *right* modifiers, such as prepositional phrases (PPs). Typically, prepositional phrase attachment is difficult to resolve locally, so Proteus does not attach PPs unless strong local evidence is present. This evidence is always semantic in nature, and must be built in as scenario- or domain-specific patterns. An example of a domain-specific NP pattern would be one which is used to capture NPs like "the president of the company".

There are some cases where PPs *can* be attached in a domain-independent manner, as in the so-called "light" nouns denoting groups of objects. Examples of these are "a team of astronauts", or "group of companies". In these cases, Proteus contains patterns to attach the PP and set the class of the resulting LF to be the head noun of the PP. That is, we want the entity corresponding to "a team of astronauts" to behave like "astronauts" rather than like "team", so subsequent patterns seeking constituents of semantic type *astronaut* would have a chance to match the entire NP. Another way to think about these light nouns is that they

behave like quantifiers, e.g., "dozens of astronauts", or "some of the astronauts".

The next set of patterns builds more complex NPs, based again on semantic information. These NPs may involve appositions, conjunctions, or subordinate clauses. Example of such higher-level NPs would be "Intel, the computer chip manufacturer", or "John Smith, 40 years old, CTO and vice president for domestic gadgets".

The highest-level pattern phase identifies scenario-specific clauses and nominalizations — the places in text where the sought events or relationships occur. This phase is the focus of this study and is discussed in greater detail in section 4.2.

In Proteus, these four modules operate by matching *patterns* of successively increasing complexity against the input. Each pattern consists of two parts:

- a *trigger* or *precondition*, which matches a certain sequence of surface sentence constituents; and
- an *action* which is performed when the precondition matches, and prescribes the operations needed to map the surface sentence fragments into semantic objects.

The actions perform operations on the *logical form* representation (LF) of the processed segments of the discourse. Thus, after these phases are complete, the discourse is represented internally as a sequence of LFs corresponding to the entities, relationships and events encountered so far in the analysis.

Each LF is a predicate with arguments; here we represent LFs as objects with named slots (see example in table 4.1). One distinguished slot in each LF, called "Class", determines the number and the types of the remaining slots that the object

Slot	**Value**
class	C-Company
name	Coca-Cola, Inc.
location	...
...	...

Table 4.1: LF for the text: "Coca-Cola, Inc."

may contain. E.g., an entity of class "Company" usually has a slot called "Name". It can also have a slot called "Location" which points to an entity describing a geo-political entity, thereby establishing a relation between the location entity and the matrix company entity. For example, the noun phase from the document in Appendix A,

```
... Information Resources Inc.'s London-based European
Information Services operation ...
```

would induce the LF in table 4.2. Note that once a pattern has identified that this

Slot	**Value**
class	C-Company
name	European Information Services
location	entity $\Longrightarrow$ <*London*>
parent	entity $\Longrightarrow$ <*Information Resources Inc.*>

Table 4.2: A complex NP and corresponding entity LF

entire text segment forms a single noun group and the LF is built, the segment is annotated with a link to the LF, and subsequent access to the text will return the

information stored in the LF, with (generally) no further reference to the originating text.

Events are more complex kinds of relations, usually syntactically headed by a verb and usually having several entities as operands. For example, the first three clauses of the demonstration text would induce the following events (the rest of the text produces no event LFs):

Slot	**Value**
class	Start-Job
company	entity $\Longrightarrow$ <*Euro.Info.Serv.*>
position	entity $\Longrightarrow$ <*President*>
person	entity $\Longrightarrow$ <*Garrick*>

Table 4.3: Event[1]: LF for "...operation has appointed GG as...president"

Slot	**Value**
class	Succeed
successor	entity $\Longrightarrow$ <*He*>
predecessor	entity $\Longrightarrow$ <*Talbot*>

Table 4.4: Event[2]: LF for "He succeeds...Talbot"

4.1.4 Logical Phases

The subsequent three phases operate on the logical forms built in the preceding pattern-matching phases. *Reference resolution* merges co-referring expressions. It links anaphoric pronouns to their antecedents, and resolves multiple mentions of the same event together. The basic mechanism behind reference resolution in Proteus

Slot	Value
class	Leave-Job
company	entity $\Longrightarrow$ <*?*>
position	entity $\Longrightarrow$ <*?*>
person	entity $\Longrightarrow$ <*Talbot*>

Table 4.5: Event[3]: LF for "...Talbot, who resigned"

is unification. Two objects are unified if their slots are compatible, where compatibility is determined by semantic type. E.g., the anaphoric 3rd person pronoun in event 2 would be resolved to the the entity corresponding to the name "Garrick", based on semantic compatibility and on the relative order of the LFs. Thus after reference resolution Event 2 would become as in table 4.6.

Slot	Value
class	Succeed
successor	entity $\Longrightarrow$ <*Garrick*>
predecessor	entity $\Longrightarrow$ <*Talbot*>

Table 4.6: Event[2]: Modified

The *Discourse analysis* module uses higher-level inference rules to build or modify event structures, where the information needed to extract a single complex fact is spread across several clauses. E.g., Proteus contains a rule of the form:

$$startJob(P_1, C, J) \wedge succeed(P_1, P_2) \Longrightarrow leaveJob(P_2, C, J)$$

The predicates in the rules are typed; P_i are variables of type *person*, C is of type *company*, and J is of type *job-post*. This rule would combine events 1 and 2 to infer

event 4 as in table 4.7, which would then unify with the underspecified event 3.

Strictly speaking, the inference rules constitute yet another scenario-specific component, which, however, is *not* yet customizable through the PET interface. For the moment, they are built into the system in the form of Lisp code. We believe, however, that if the structure of the output is defined appropriately (viz. it is designed to correspond closely to events as they appear at the surface sentence level), the need for inference can be minimized. Where the output templates do not correspond closely to the surface structure, we believe that the inference operations can be successfully modeled through a straightforward command interpreter, by means of commands stated in a relational algebra (e.g. SQL-like statements).

In the *output generation* phase, Proteus selects which of the resultant LFs are to be output[1], and formats them into the output structure specified by the user, e.g., into a database table.

Slot	**Value**
class	Leave-Job
company	entity $\Longrightarrow$ <*Euro.Info.Serv.*>
position	entity $\Longrightarrow$ <*President*>
person	entity $\Longrightarrow$ <*Talbot*>

Table 4.7: Event[4]: Inferred Event

A thorough example of extraction which illustrates the operation of the various phases of Proteus is given in [19].

[1]Some LFs are *internal*, such as Event 2. They are used for intermediate inference, and are not presented on output. The form of the output is given by the task specifications.

4.2 Organization of the Pattern Base

Before we describe our example-based strategy for pattern building, we examine the organization of the pattern base in more detail. The patterns are arranged in layers according to their range of applicability:

1. *Domain-independent*: this layer contains the most general patterns. E.g., the patterns for name recognition (for people, organizations, and locations, as well as temporal and numeric expressions, currencies, etc.), and the purely syntactic patterns for noun and verb groups. These patterns are useful in a wide range of tasks.

2. *Domain-specific*: the next layer contains domain-specific patterns, which are useful across a narrower range of scenarios, but still have considerable generality. These include domain-specific name patterns, such as those of certain types of artifacts, as well as patterns for noun phrases which express *relationships* among entities, such as those between persons and organizations.

3. *Scenario-specific*: the last layer contains scenario-specific patterns, having the narrowest utility, such as the clausal patterns that capture relevant events.

This stratification reflects also the relative "persistence" of the patterns. The patterns at the lowest level have the widest applicability. They are built in as a core component of the system, because these are not expected to change when the system is ported to a new domain. The mid-range patterns, applicable in certain commonly encountered domains, can be organized into domain-specific *pattern libraries*, which can be plugged in as required by the task. For example, Proteus has a set of low-level, domain-independent patterns for extracting entities, such as:

- organization,
- company,
- person,
- location;

For the domain of "business/economic news", these entities are used to build a library of patterns to capture domain-specific relations:

- person—organization,
- organization—location,
- parent—subsidiary organization.

The scenario-specific patterns must be built on a per-scenario basis. Because, in moving to a new scenario, we expect the highest turn-over patterns at this level, we invested greatest effort in making this step easier for the designers of IE systems. Our first attempt to do so was PET, described in the following chapter.

4.3 Formal Evaluation

The Proteus core IE engine takes the original text as input and produces SGML output. The original text usually already contains some form of SGML/XML mark-up, e.g., to identify titles of articles, headlines, dates, paragraph and document boundaries, etc. For the NE task, the result of extraction is a modified copy of the original document in which the NE tags are inserted as additional SGML mark-up. For the ST task, the output is a report file containing the extracted templates, in an

object-like format. Sample output files for these two tasks are shown in Appendix B and C, respectively.

When correct answer keys are available, the system's response can be matched against them by the scoring program. This allows the user to evaluate the system's performance on the given document, by comparing the results against predefined answer keys.

The measures used for IE evaluation are similar to those used for IR, i.e., recall and precision. In IR, where the goal is to retrieve good quality documents from a text collection, recall and precision are defined as follows, in terms of three values:

$$R = \frac{C}{C + M} \tag{4.1}$$

and

$$P = \frac{C}{C + S} \tag{4.2}$$

where C is the number of documents correctly retrieved, M is the number of missing documents, which the system should have retrieved but did not, and S is the number of spurious documents, those that it should not have retrieved, but did.

In IE the situation is a bit more complex, since we are measuring slots filled in the semantic objects that the system found in text. Thus in the case of IE, there is one more category of error: an object or a slot can be identified *incorrectly*, i.e., the system should have indeed picked up some object from a given place in text, but it picked up the wrong object. If we denote the number of incorrectly identified slots or objects by I, the recall/precision formulas become:

$$R = \frac{C}{C + I + M}$$

and

$$P = \frac{C}{C + I + S}$$

Sometimes it is more convenient to measure performance with a single number; in such cases F-measure is used, which is a weighted geometric mean of the two:

$$F = \frac{(\beta + 1)RP}{(\beta * R + P)}$$

where β is a parameter controlling the relative importance of recall and precision to the evaluator. In all our experiments, we use $\beta = 1$.

A complete evaluation requires answer keys. For the MUC-6 competition, DARPA generated a set of answer keys for a training corpus and a test corpus, each consisting of 100 articles from the Wall Street Journal (WSJ).

4.4 Problems in Scoring IE

We should note that the MUC evaluation is itself quite a complex conceptual and algorithmic problem. In particular, it is much more complex than scoring recall and precision for IR tasks. The additional complexity arises from the fact that a given document is not simply relevant or irrelevant. A document may induce more than one event and each event typically has several slots—and each slot leaves potential room for error.

In general, the answer keys, and the system response, may contain more than one event. The situation is complicated by two factors:

1. the events in the response may contain errors, in that some events may be spurious and some slots may be filled incorrectly, and

2. the two sets of events form *un-ordered* sets.

To score the response against the answer key, the scorer must first *align* the two sets of events, and then, to appear impartial, it must try its best to give the highest possible score. At the same time, the lack of order is inherent to the situation, because although the original text is linear, in general it is difficult to anchor a given event at a particular segment of text, since an event may be pieced together by means of complex inference from *several* partial cues or descriptions.

The search for an optimal alignment causes a serious problem. Even foregoing the fact that it may require checking all possibilities, it causes a serious problem of "deceptive scores". In effect it is possible for one IE system (or one version of a system), S_1, to be "objectively" worse than another, S_2, while the scorer assigns a higher score to S_1 than to S_2.

This has gone down in MUC lore as the "Jesuit priest" phenomenon. In the context of the MUC-3/4 topic, terrorism in Latin America, the developers of Proteus also carefully engineered a second system (each participating site is allowed to submit more than one system for evaluation) which religiously reported that "six Jesuit priests were killed in El Salvador" for *every* document, disregarding the input. At the formal evaluation, this rogue system received a higher score than some of the competing systems, which actually tried to perform meaningful extraction.

Here is a simple example of how this can happen. Suppose the correct answer contains a single event, $e^{\uparrow}$, and suppose also that S_1 reports an event $e_{\downarrow}$, which is basically wrong—but, say, some of the slots happen to be correctly filled. The scorer will assign *some* credit to $e_{\downarrow}$ based on the fact that *some* event was found and some slots were correctly filled.

Now, for the system to improve *objectively*, two things must happen: it must learn

a. to discover the correct event $e^{\uparrow}$, and

b. to stop discovering the incorrect event $e_{\downarrow}$.

If S_2 satisfies both **a.** and **b.**—i.e., $e^{\uparrow}$ replaces $e_{\downarrow}$ in the response—it will clearly get a higher score than will S_1. However, if only one of **a.** or **b.** takes effect, the score for S_2 can actually go down: in case of **a.** it will be penalized for a spurious event ($e_{\downarrow}$), and in case of **b.** for a missing one ($e^{\uparrow}$).

Note that, in general, *either* only **a.** or **b.** will happen as an intermediate state, during a normal development cycle, since, in general, two different system components are responsible for finding $e^{\uparrow}$ and $e_{\downarrow}$, and these components are debugged separately. So that, in general, it is not a simple matter of tweaking a single component to find $e^{\uparrow}$ instead of $e_{\downarrow}$.

There have been several proposals on how this problem may be fixed or alleviated. It has been suggested that the scoring algorithms be extended with some notion of relative *importance* of slots inside the events. So that events receive points only if certain constraints on important slots are satisfied. The situation is further complicated by the fact that recent evaluations included scoring of *extents*, i.e., assignment of credit for identifying the precise spans of text which induce the reported events.

Scoring for IE remains a topic for further research. Despite the problems described here, these methods are the standard way of measuring quality in IE, and we resort to them in our own evaluation, in chapter 8.

Chapter 5

PET: Example-Driven Acquisition

PET is a set of graphical tools for customizing knowledge bases for a new scenario. Figure 5.1 shows the principal components of the toolkit and their interaction with

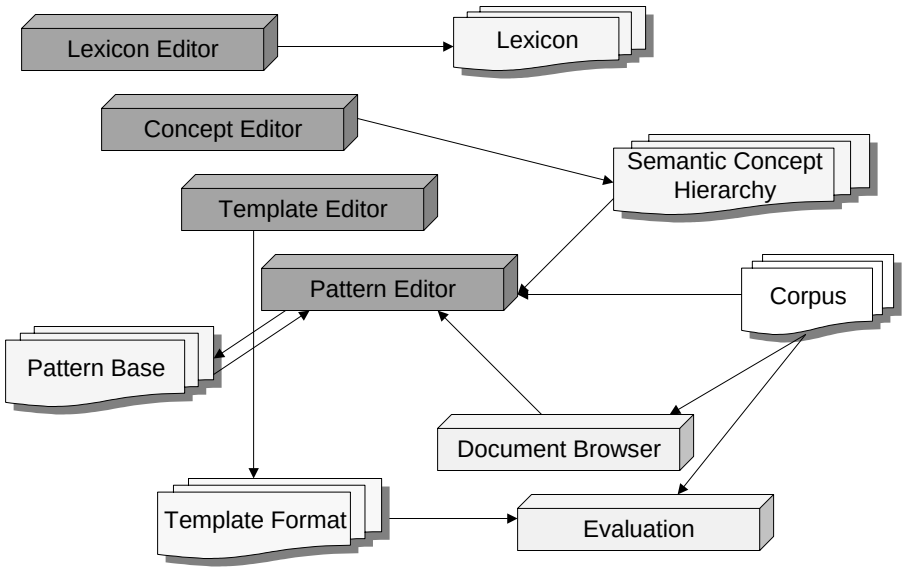

Figure 5.1: PET components

the KBs of the preceding chapter.

The modules of PET at present are organized into three principal groups, represented by the boxes laid out along the diagonal:

- Document Browser: for performing various IE operations on document texts;
- Knowledge Base Editors: for customizing the Proteus knowledge bases;
- Evaluation tools: for testing and applying the acquired KBs to a training corpus.

We now introduce these tools in turn. Together they comprise a prototype of an integrated graphical environment, intended to support the development cycle of scenario-level customization. The tools are written in Common Lisp (Allegro); this allows seamless integration with the core Proteus system, also written in Lisp. The graphical package used was Garnet [15], developed at CMU.

The user begins the interaction by bringing up the PET control panel. From the main panel the user can invoke the various components of PET, which sits "on top of" Proteus, and in turn invokes its functions.

5.1 Document Browser

We start with the description of the Document Browser, since that is where the development process typically begins—with an actual document to be analyzed. This tool allows the user to:

- Select and browse documents from a training collection.
- Perform various IE tasks on the selected document.

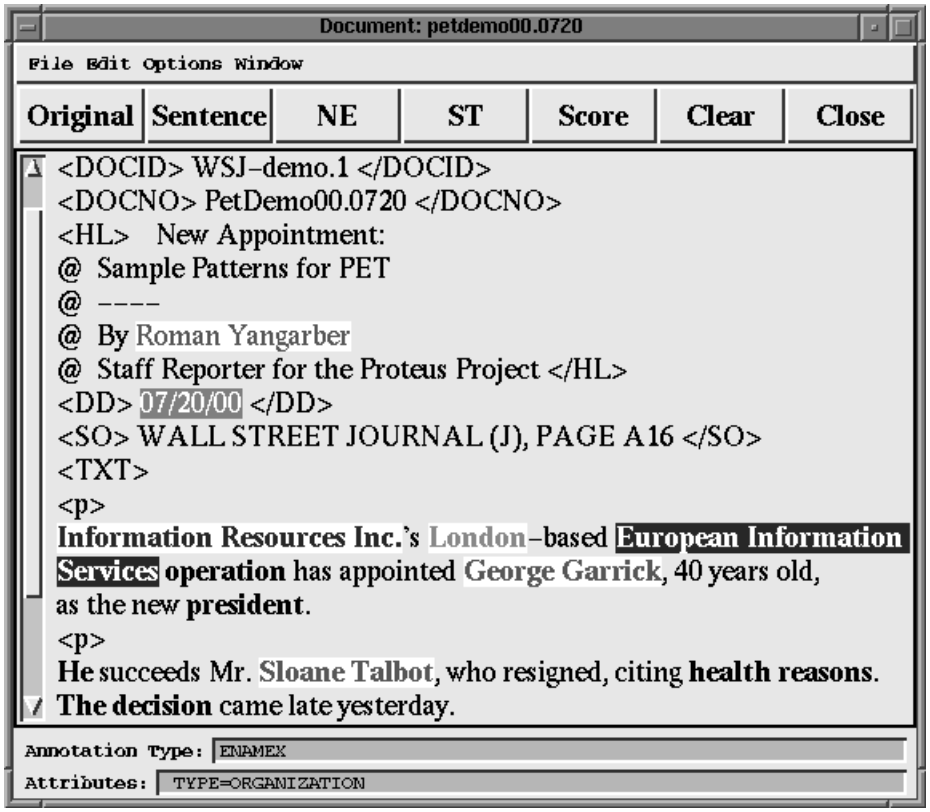

Figure 5.2: Main Document Browser, with NE Results

- View the *results* of IE in graphical form.

Figure 5.2 shows the document window, containing the demonstration text from Appendix A. We will use this as a *pilot* example in the present chapter to illustrate PET's functionality.

At present PET supports the following MUC tasks:

NE – Named Entity,

ST – Scenario Template.

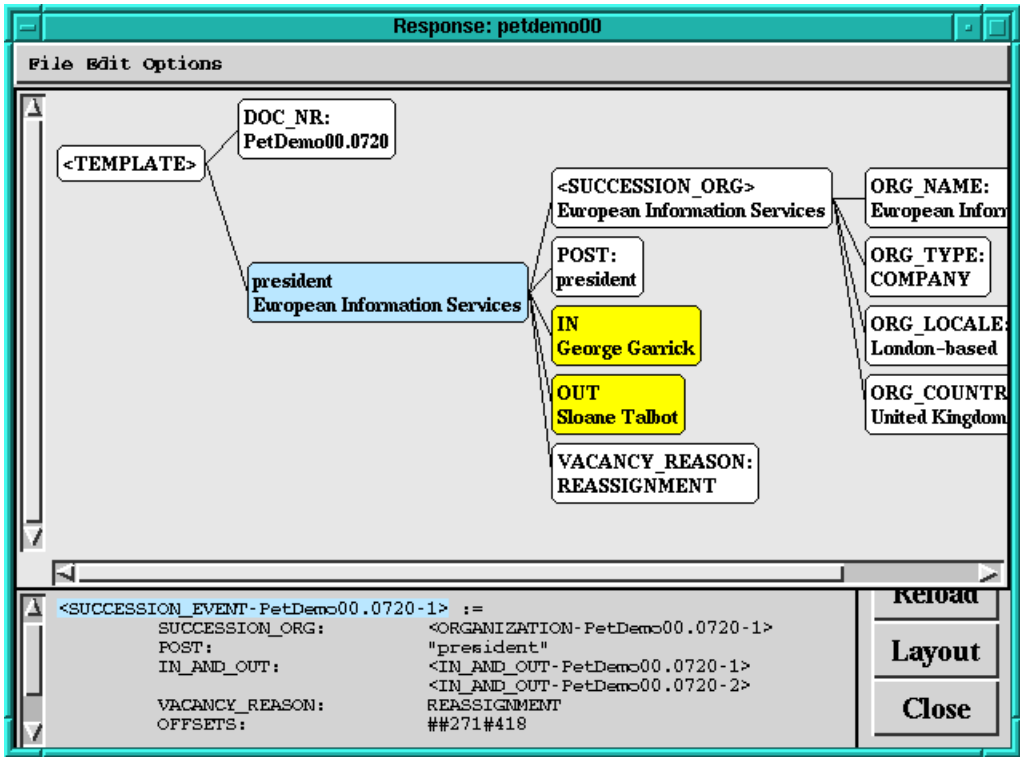

Figure 5.3: ST Results

The output Proteus produces for the NE and ST tasks is rather opaque (cf. Appendices B and C), and imposes a considerable processing load on the human reader. The document browser presents the results in an intuitive graphical form.

The results of NE appear in figure 5.2, with named entities in the various categories color-coded: persons, locations, organizations, dates, currencies, etc. Additionally, un-named entities—general noun phrases—are shown in boldface. When the user clicks on an entity in the text, the message panel at the bottom of the main window displays the type of SGML annotation corresponding to the entity, in standard MUC format.

Figure 5.3 shows the results of ST processing on the same document. Again, the graph is intended as an aid to viewing the system output (Appendix C). The reader can readily appreciate the reduction in cognitive load by comparing the two representations. The results are shown as a graph in which each node corresponds to a slot in an extracted object. The bottom panel shows the last-clicked object in the MUC standard format—object name followed by its slots given as name-value pairs. The last clicked node was the one labeled "president/European Information Services."

We should briefly mention a peculiarity of the output format used for MUC-6 output. In this scenario, the top-level output structure is the so-called *"succession_event"*. According to MUC-6 task specifications, this structure is intended to group together *all* transitions pertaining to a given post in a given company. Proteus's internal representation stores each transition event as a separate record in a table of events. In relational terms, this simply means that Proteus's transition event table, containing the actual hiring/firing events, is not the final output form. Rather, the output form is a particular *view* on this table. The unique key in the internal table is the pair `(company, post)`, and the output view has a field called `in_and_out`, which contains a *set* of transition events matching a given key. In effect, this organization of the MUC output format introduces an extra level in the response graph.

The graph in the figure shows that the pilot document induced a *single* succession event in which one person left the office and another took over. This transition is registered by having two elements in the `in_and_out` slot.

Slot values can be atoms, as, e.g., the slots called `post` and `vacancy_reason`, or they can themselves contain nested objects, as does the slot `in_and_out`. Initially, the graph shows only the top-level nodes. The nodes at the deeper levels can be exposed or hidden by clicking on the parent nodes.

The last clicked node is shown in blue; nodes that can be expanded further are colored yellow, and nodes that cannot be expanded ("leaf" or *atomic* slots) are white.

PET's Document Browser includes a similar window, which allows the user to view the *answer keys*—when such are available—in the same graphical form, simultaneously, while reviewing the system response. This is convenient for comparing the system output against the answer keys.

Both types of graphs, the system response and the answer key, have links back into the source document; clicking on a node in the graph highlights the corresponding text segment(s) in the main document window.

The layout of the ST graphs can be rearranged by the user, by clicking and dragging the nodes. As discussed in section 4.4, *aligning* the events in the system response with those in the answer key is a non-trivial problem. This tool provides aid to the user for visualizing possible alignments.

5.2 Knowledge Base Editors

Prior to MUC-6, the knowledge bases were coded in Lisp as part of the Proteus system. The KBs were modular only insofar as they resided in separate files, but there were no uniform convenient facilities for modifying the KBs "on the fly". Generally, for a modification to a KB to take effect, the corresponding segment of

Lisp code had to be explicitly evaluated, and in some cases, the entire system had to be re-loaded.

With the advent of PET, we developed new file formats for storing the KBs on disk, to facilitate dynamic modification of knowledge bases. The following KB editors have been implemented to date:

- *Lexicon Editor*: Allows the user to build domain-specific dictionaries. These dictionaries are used to supplement the COMLEX dictionary, which provides the base English lexicon, [20, 31].

- *Concept Editor*: Lexical entries are grouped into a hierarchy of semantic classes, similarly to, e.g., WordNet, [33]. Pattern elements can then refer to these concept classes. The editor allows the user to organize classes in a directed acyclic graph (DAG), and to examine and modify the domain-specific hierarchy.

- *Predicate Editor*: Lets the user define predicates—the internal logical structures which are filled when patterns match. The editor also allows the user to specify how the predicate is represented on the output, and how it interacts with the pattern editor.

- *Pattern Editor*: The IE engine operates by applying sets of semantic patterns to the text sentences. The pattern editor allows the system to acquire the patterns from example sentences provided by the user. The user also specifies—with help from the system—the structure which should result from the example sentences. It also enables the user to control the order in which patterns are applied and form groups of "competing" patterns.

As we mentioned earlier, the customization process is logically centered around the pattern base. We now turn to a more in-depth customization example, which will occasion a description of each of the KB editors.

5.3 Editing Patterns

The main objective of PET is to engage the user at the level of surface representations as much as possible, while hiding the internal operation of Proteus. When adding patterns, it is possible for the user to restrict her/his input to

- providing *examples* of text which contain events of interest,
- describing the corresponding *output structures* (LFs) which the example text should induce.

We will describe how the system uses this information to

- automatically build patterns to map the user-specified text into the user-specified LF,
- *generalize* the newly created patterns to boost coverage; the generalization happens on both the syntactic and the semantic levels.

Suppose the developer has found a salient text segment and intends to tune the IE system to extract the information from it. Figure 5.4 shows a segment from the pilot text, with the extracted event, in the form of a database record.

In the era of pre-PET Proteus, upon finding a candidate example, the developer had to construct a pattern to capture the example and extract an event from it. As we observed in chapter 4.1.3, the pattern consists of a trigger and an action. In

... Information Resources Inc.'s London-based European Information Services operation has appointed George Garrick, 40 years old, as the new president ...

Field	**Value**
Position	president
Company	European Information Services
Location	London
Person	George Garrick
Status	In

Figure 5.4: Succession text and extracted record

this case, the trigger should match an active clause beginning with a noun phrase, *np*, of type "company", followed by a verb group (*vg*) of class "appoint", followed by a *np* of class "person", etc. The action should contain instructions which specify how to assemble the matched constituents into an event structure.

Figure 5.5 shows an excerpt from the pattern code. This code is intended to instill dread and gloom in the aspiring IE developer; it should be apparent that this method of development is time-consuming and error-prone.

Instead, PET employs a kind of interactive "bootstrapping": it takes a new example from the user, and applies to it the patterns that the system has acquired *so far*. This typically produces a partial analysis and builds LFs for the analyzable constituents of the example text. The user then specifies how these LFs, or their sub-parts, combine to yield the LF for the entire example.

```
;;; For <company> appoints <person> <position>
(defineTrigger Appoint
   "np(C-company)?  vg(C-appoint) np(C-person)
           to-be?  np(c-post):
    company=1.attributes, person=3.attributes,
           position=5.attributes |
 ...
```

```
(defineAction Appoint (phrase-type)
    (let ((person-at (binding 'person))
       (company-entity (entity-bound 'company))
       (person-entity (entity-bound 'person))
       (position-entity (entity-bound 'position))
       new-event)
     ;; if no company slot in position, use agent
 ...
```

Figure 5.5: A manually coded scenario pattern

5.4 Acquiring Preconditions

To illustrate the operation of PET, we first show how a user would use the system to acquire a *clausal* pattern from the example in figure 5.4. Consider figure 5.6, which shows the main window of the pattern editor.

The top window of the editor is the *example* window. It contains the text of the original example—typed in by the user or pasted from the document browser. When the user clicks the "Match" button, PET invokes Proteus to analyze the example. The main point here is that the basic system, i.e., one that has not been specialized for any particular scenario, will produce a *partial* analysis, by dint of

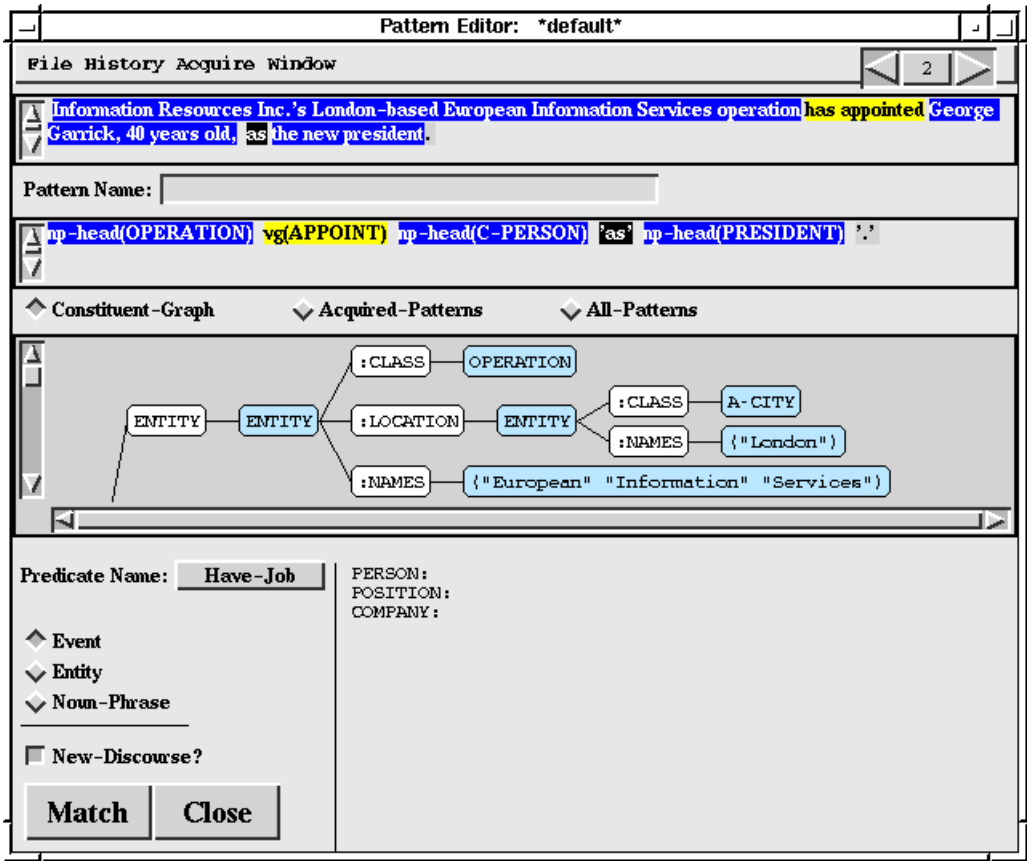

Figure 5.6: Initial analysis

the built-in patterns for named entities, and shallow syntactic patterns for noun groups and verb groups.

Below the example window is the *precondition* (or trigger) window. It contains the initial analysis, the precondition proposed by the system. This is the first approximation to the precondition of the pattern that the user will ultimately accept.

The initial analysis indicates that Proteus found six separate segments in the sentence. The sentence begins with a long noun group, whose head is the lexical

item "operation". (This is picked up by a complex noun group pattern, discussed below in section 5.8.) Following that, Proteus finds a verb group, followed by another NP, followed by a literal "as", which cannot be analyzed as part of any constituent, and so on. The corresponding parts between the two windows are color coded, with NPs in blue, verb groups in yellow, literals in black, etc.

Proteus builds the underlying syntactic structures for each constituent identified in the precondition window, as explained in section 4.1.3. When the user clicks on a constituent, its structure is displayed in the form of a graph in the third window of the pattern editor, the *constituent* window. If the constituent is a noun phrase, Proteus builds the corresponding semantic structure as its logical form (LF), i.e., the corresponding *entity*, which is displayed together with the syntactic information.

In this case, the user clicked the first constituent, and the graph of its entity shows its class as "Operation", and the fills for the slots `:names` and `:location`. Note that the value of the `:location` slot is a complete nested entity.

At this point, the system engages the user in an interaction, in which s/he can operate on each constituent. The menu of operations includes generalizing the constituent, applying regular expression operators such as optionality or closure, removing it, adding user-defined pieces to the pattern, etc.

There is no one "correct" way to transform an example into a pattern; doing so requires some basic linguistic awareness on the part of the user, as well as an understanding of the subject domain. Typically the user needs to find an appropriate level of *generalization* for each NP or VG constituent, to maximize coverage without jeopardizing precision.

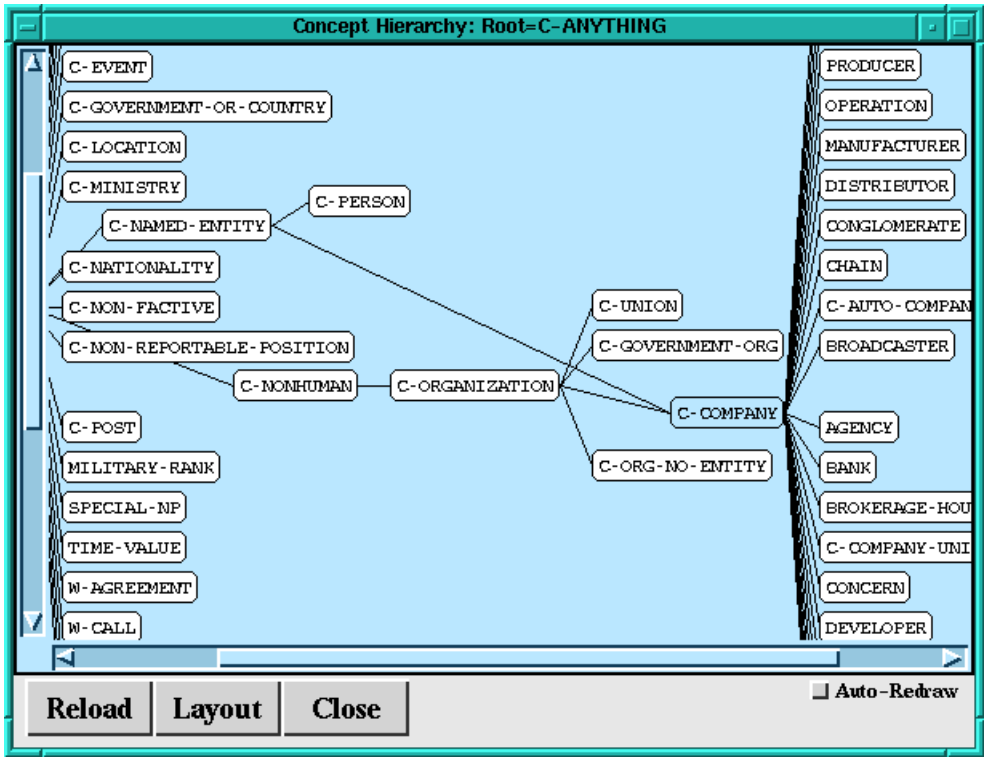

Figure 5.7: Concept Editor

5.5 Semantic Generalization and the Concept Editor

Consider the first constituent in the precondition, `np_head(OPERATION)`. It would probably serve our purpose best—the purpose being broader coverage for the pattern — if the first noun phrase were generalized to include terms denoting different types of businesses. The pattern should match any semantically similar noun phrase in that position in the clause: "concern", "outfit", etc. To this end, PET/Proteus allows the user to gather semantic concepts in an inheritance hierarchy, outlined above in section 5.2. For example, s/he can gather all these and more lexemes under the one semantic class, called *C-Company*. PET then allows the user to perform

semantic generalization on the individual constituents of the pattern's precondition.

In the generalization process the user employs the concept hierarchy, which can be viewed and modified through the Concept Editor, figure 5.7. The editor allows nodes in the graph to be rearranged, items to be added, and modified.

The semantic hierarchy is scenario-specific. It is built upon a domain-independent base, which includes some general concepts like persons, locations and companies. Then for each specific scenario, the user adds items dynamically, either as relevant terms are found in training corpora, or through a pair of auxiliary external tools that provide a link to

- WordNet, the pre-existing domain-independent hierarchy, and
- tables of domain-specific word similarities, based on co-occurrence statistics, [12].

The parent class of the node "Operation" (upper-right corner) is "C-Company" and it includes many related terms. This class also includes unspecified types of businesses identified only by a *proper name*—a name which was determined to refer to a company by low-level name identification patterns. Such low-level patterns create entities and set their class to "C-Company".

Right-clicking on a constituent brings up a menu of operations. The user can choose to generalize the *class* of the NP up the hierarchy one level at a time. Again, the actual candidates for the classes are suggested by the user's intuition, by WordNet, or by co-occurrence statistics.

Likewise, the second constituent should be generalized to a class of all verbs similar to "appoint", such as "name", "nominate", and "hire". The fifth NP should

be generalized to a semantic class that includes all corporate titles; in Proteus, this class is called "C-Post".

The literal 'as' should be marked optional to match sentences like "John was appointed president".

Lastly, the final period, which appeared as a literal constituent, should be removed, since otherwise we would have the unnatural restriction that this clause match only at the *end* of a sentence.

These operations result in the following pre-condition:

```
np-head(C-COMPANY) vg(C-APPOINT) np-head(C-PERSON) 'as'?
      np-head(C-POST)
```

PET allows the user to state more complex constraints on the constituents. What we have seen above are simple constraints on the `:class` slot which state that the value in that slot must be at or below some specific node in the concept hierarchy. The constraint `np_head(OPERATION)` is a shorthand for the more general expression: `[<.entity.class, OPERATION>]`. The general form with brackets can express a conjunction of multiple constraints, on several different slots of an entity simultaneously, each constraint providing for different levels of specificity in the concept hierarchy. Thus, for example, it is possible to state that the `:names` slot must be non-nil, or some other slot have a certain value, or that the class of that value be exactly at the specified node of the hierarchy, or above it, or below.

These mechanisms provide the finer control in pattern building for the advanced user. PET can be driven both as an automatic and as a stick shift. These and still more esoteric details are documented in a user manual, [54].

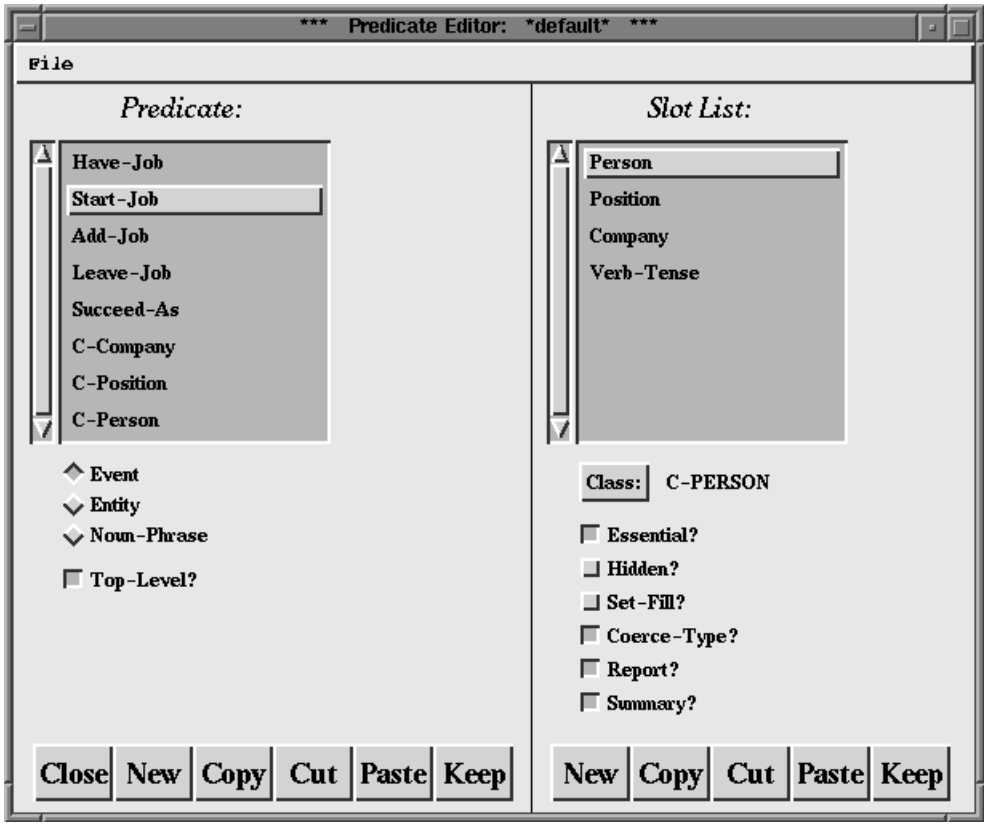

Figure 5.8: Predicate Editor

5.6 Acquiring Actions and the Predicate Editor

Now the user specifies what action is to be performed when the precondition matches. S/he first selects the class of LF to be created. The type of LF is chosen from a list of *predicates* stored in a separate knowledge base, of which some are events and others entities. Figure 5.8 shows a sample predicate base for the MUC-6 task.

The predicate editor allows the user to define new types of LFs and to specify what slots they contain. For each slot, the user can set several parameters, indicat-

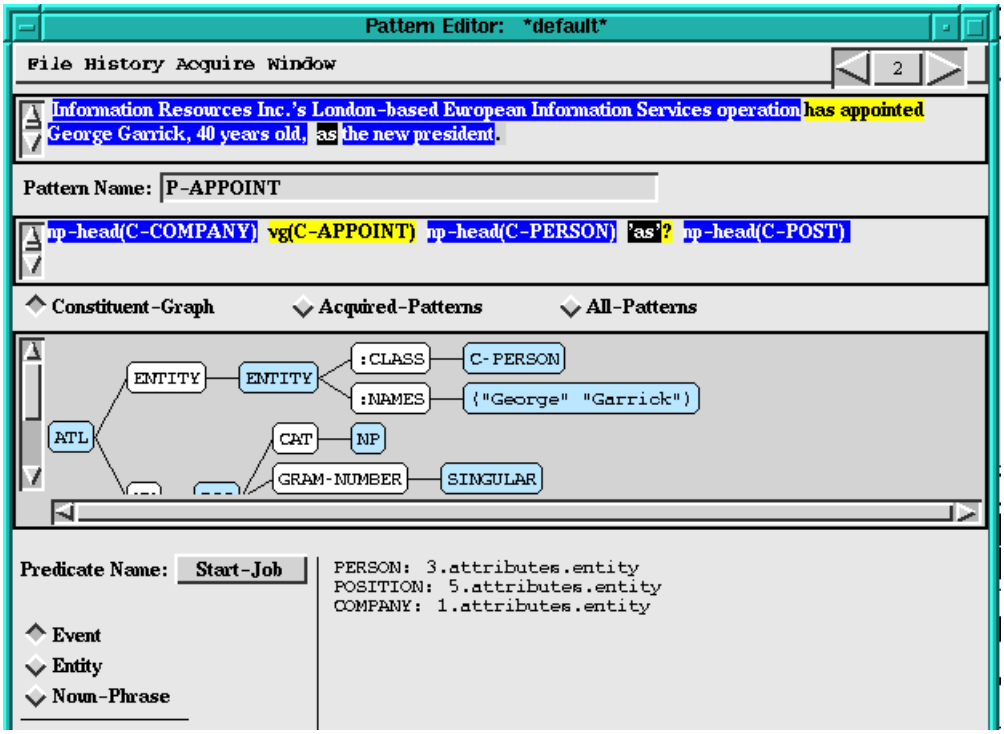

Figure 5.9: Final Pattern

ing what type of object the slot can receive, whether the slot is hidden—or shown on the pattern editor—whether and how it appears in the final output, etc.

In our example, the appropriate predicate is `Start-Job`. The name of this predicate is chosen from the drop-down menu on the pattern editor in figure 5.6, and the slots pertaining to the predicate appear next to it on the right.

The user now indicates how the constituents (or, more precisely, the LFs) found in the example are to function as arguments of the new event. This is done by dragging-and-dropping nodes in the constituent graph into the predicate slots. The node which is dragged identifies the sub-structure of the constituent that is used to fill the slot.

The pattern definition is complete, figure 5.9. The precondition and the action together constitute a full-fledged pattern which matches a clause and produces a LF. The user provides a name in the text box below the example window (labeled "Pattern Name"), and instructs PET to *accept* the new pattern.

5.7 Meta-rules: Syntactic Generalization

Consider the following variant of the original example:

> ... Jane Smith, *an avowed anti-capitalist,* was named *yesterday* as the next CEO of IBM, Corp., ...

The basic pattern for an active clause, which we acquired in the preceding section, will not match this passive clause. There are two essential kinds of variation here:

- *syntactic transformation*: the system should have syntactic variants of the original pattern, to capture the corresponding passive clause, relative clause, and others.
- optional *modifiers*, which are not semantically relevant to the scenario, but may intervene randomly between pattern constituents. Such modifiers may be arbitrary sentence adjuncts, appositions and PPs, as exemplified by the italicized segments above.

How do we capture the syntactic paraphrases of the original pattern? The user could, of course, provide *transformed* examples, build patterns individually for each transformation, and insert the optional modifiers to make the patterns as general as

possible. This naive approach would quickly lead to gross proliferation of patterns, rendering the user's task unmanageable.

Instead, PET invokes Proteus's *meta-rule* mechanism: after a pattern is accepted, the system can generate all related generalizations of the pattern *automatically*.[1] For example, from the active clause pattern above, a passivizing meta-rule will generate the pattern:

```
np-head(C-person) RN? SA? pass-vp(C-APPOINT) SA? 'as'?
      np-head(C-POST) ['by' np-head(C-company)]?
```

The modifiers are made optional by the '?' pattern operator. The first modifier, `RN`, is a built-in syntactic *sub-pattern*, a macro whose name may be used inside another pattern. `RN` matches any right noun-phrase modifier—a prepositional phrase, an apposition, or a subordinate clause, delimited by commas; `SA` is a sentence adjunct, and *pass-vp* is a passive VP.

The resulting pattern will match the passive example, and will produce the correct event LF. To maximize coverage, the system should contain meta-rules for all clausal variants, including nominalizations. There is on-going work on building mappings from verb roles to nominalization arguments, [32]. Meta-rules can also be provided to generalize noun-phrase patterns, which are discussed in the following section.

[1] A similar meta-rule mechanism is also featured in the SRI FASTUS system[3].

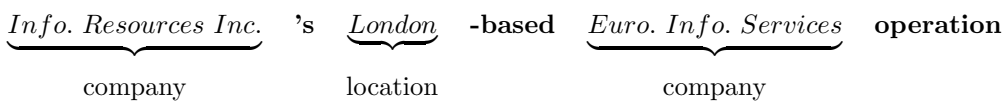

Figure 5.10: Initial Analysis of NP

5.8 Acquiring Lower-level Patterns

By a similar process, we can acquire the pattern for analyzing the portion of our example shown in table 4.2. This is a complex NP made up of a series of nouns, names, and other lexical items.

The basic system would analyze this example as in figure 5.10.

and propose the precondition:

```
n(C-Company) 's n(C-City) -based n(C-Company) n(operation)
```

The literal constituents are shown in boldface.

The approach described here involves the creation of a semantically-specific noun group pattern which will take precedence over the general noun group pattern. In consequence, the general noun group pattern is not applied in analyzing the example. The customization tool also supports an alternative mechanism for special noun groups, whereby they are first analyzed by the general noun group pattern and the resulting LF is then restructured by a semantically-specific pattern. The analysis also creates LF entities for each segment italicized in figure 5.10.[2]

The user proceeds to complete the precondition of the pattern as:

[2]For purposes of presentation, we have simplified the form of these patterns to emphasize the analogy with the clause patterns. In the current implementation, each pattern element would involve a conjunction of tests for the syntactic type (n) and the semantic class (*C-Company*, etc.).

```
[n(C-company) 's]?  [n(C-city) -based]?  n(C-company)
      n(operation)?
```
[3]

To acquire the action, the user can designate one of the generated entities as the "head" entity, or the *matrix* entity, for the complex NP. In the example, we may designate the fifth constituent, `n(C-Company)` as the matrix, which corresponds to the main company's name. The remaining entities are *subordinate* to the matrix, i.e., as standing in some semantic relation to it. The first constituent (when present) fills the "parent" slot, and the third constituent fills the "location" slot. To establish the relation, the user drags-and-drops the subordinate entity into the appropriate slot in the matrix entity (see figure 4.2).

5.9 Evaluation Tools

The *Evaluation tool* allows the user to test the effect of the acquired KBs on system performance. This can be done by:

- Processing and reporting scores for individual messages within the training corpus.
- Processing a complete training corpus and scoring the performance of the system.
- Estimating improvements in overall scores based on anticipated improvements in particular individual messages.

[3]Un-named sub-patterns, shown here as bracketed sequences of pattern elements, is a feature that remains to be implemented in PET/Proteus. In the current implementation, they actually need to be build in a separate step, as *named* sub-patterns.

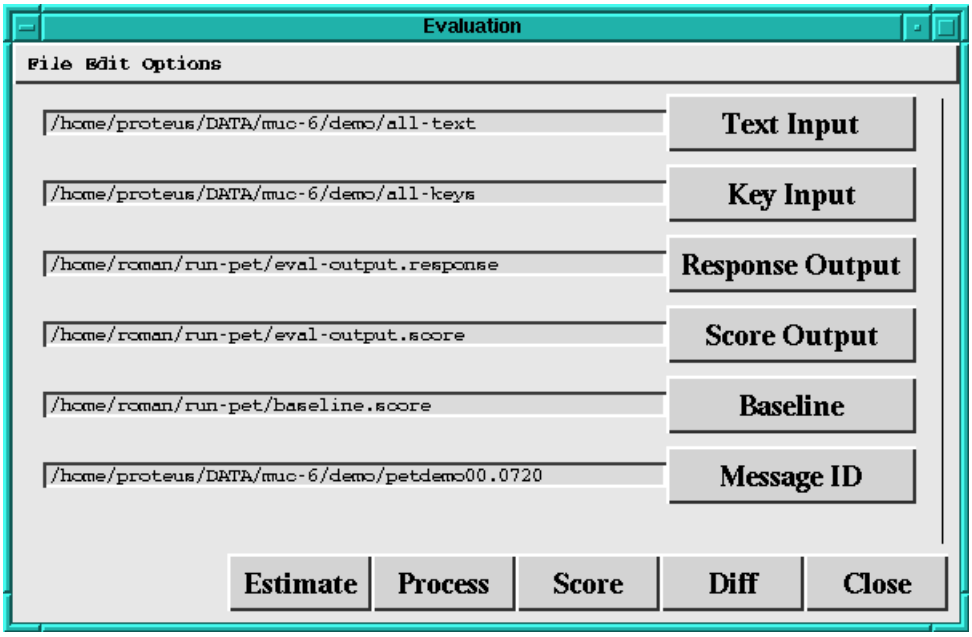

Figure 5.11: Evaluation Window

This feature allows the user to quantify answers to questions like: "how much would the overall system performance improve (in terms of the ultimate F-measure) if a particular feature were added, or a particular problem were fixed". These measures are useful for planning the development process of the IE system.

- Comparing the score of the system against a previously established baseline, giving a document by document *diff*-style report, stating which scores went up and which went down, and by how much.

The last feature (implemented in Perl) is essential in the development process, as, in general, improvements in one part of the system may lead to degradation in another. It is common that a new (imperfect) pattern, which improves recall in

one message, degrades precision in another. This tool enables the user to closely monitor these fluctuations during the development cycle.

5.10 Approximate Matching and Clausal Modifiers

The toolkit remains incomplete in several aspects. One shortcoming is that the bootstrapping process lacks a sense of "near hit". If the user acquires a pattern, and later gives an example which is very similar to the pattern but does not quite match, PET gives no indication whatsoever of the proximity. Such an indication would be very valuable, and would put PET more in line with the active learning approaches mentioned in section 2.3.3.

A more basic shortcoming of PET is that even if the user notices that two examples induce very similar patterns, or variations of the same pattern, there is no way to combine examples or patterns. PET maps examples to patterns in a strictly one-to-one fashion.

The problem is that different examples (supporting the same pattern) will typically bring in different modifiers, such as PPs in sentence adjunct positions (SAs). These modifiers should *all* be accommodated in the resulting pattern, for maximal generality. The modifiers are in general *optional* and *un-ordered*, in the sense that they may appear in any SA position, and in any combination.

For example, suppose we are building a system for the "Corporate Acquisitions" scenario, and encounter a basic pattern like

(Pattern 5.1): `np(C-Company) vg(buy) np(C-Company)`

Different examples may support this pattern, and they may contain any subset of

the following SAs, in any order:

- `'from' np-head(C-Company)` $\longrightarrow$ *Seller*
- { `'for', 'at'` } `np-head(C-Currency)` $\longrightarrow$ *Price*
- `'on' np-head(C-Date)` $\longrightarrow$ *Date*
- `'valued' 'at' 'about'? np-head(C-Currency)` $\longrightarrow$ *Price*

The first three modifiers are PPs, and the last is a subordinate clause (reduced relative passive). Each `np-head` constituent matches a noun phrase; the arrow indicates which slot in the "Purchase" predicate is filled by the corresponding NP. (The slot name is italicized.) Note that these adjuncts fill predicate slots that are *essential*, as opposed to other, irrelevant sentence-adjunct PPs, not important for the scenario,—e.g., "after a big lunch".

A naive approach with PET would require manually pasting the modifiers into *a single* example and then generating a pattern from it. This is artificial, overly complex and increases the possibility for human error. This indicates the need to extend the functionality of the pattern editor to collapse multiple examples into a single general pattern *automatically*.

In our recent work [26], we have instituted an informal procedure for collapsing multiple patterns. This involves manually "growing" a single, special sub-pattern (cf. 5.7) which contains an un-ordered *disjunction* of all the modifiers that can possibly occur with the given pattern[4]:

`SA :=`

[4]NB.: though a sub-pattern can appear only inside another pattern, it may perfectly well fill slots in the predicate of the containing pattern.

```
   user-Defined-SA*   | system-Defined-SA


user-Defined-SA :=
      'from' np-head(C-Company)
   |  { 'for' | 'at' } np-head(C-Currency)
   |  'on' np-head(C-Date)
   |  'valued' 'at' 'about'? np-head(C-Currency)
```

We have done this for several scenarios studied under PET, such as "Natural Disasters" and "Mergers and Acquisitions". While we have some experience with building patterns in this fashion, it remains to be implemented formally and in a disciplined way.

Even if this approach were formalized in PET it would not solve the problem completely. It is not sufficient to have a *single* user-defined modifier for the entire scenario. To illustrate the difficulty, suppose we now try to build up the pattern

(Pattern 5.2): `np(C-Company) vg(sell) np(C-Company)`

Supporting examples may likely contain the SA:

- `'to' np-head(C-Company)` $\longrightarrow$ *Buyer*

In our current, *ad hoc* scheme, we add this modifier to the user-defined SA above, even though this modifier can never apply to pattern 5.1. For proper generality, the user-defined SA should be able to apply to a particular pattern or verb, rather than to the entire scenario, as it does now.

5.11 Assessment

Although the issues in the preceding section remain to be addressed, at the time of writing of this thesis, the development of PET has resulted in a functioning prototype. This prototype has been used in customizing Proteus, with a variable degree of success, for several scenarios:

- "airplane crashes" (MUC-7 dry-run task),
- "missile and rocket launches" (MUC-7 formal task),
- "election campaigns",
- "effects of environmental pollution",
- "corporate mergers and acquisitions",
- "natural disasters".

The customization was performed by five graduate students in linguistics, four of whom were not system experts.

PET has been ported to Japanese and Swedish, resulting in IE systems for the "management succession" scenario.

As mentioned, PET was used for developing the MUC-7 scenarios for our official MUC-7 system, [56]. Without delving into the complexities and idiosyncrasies of the MUC-7 scenarios, we should mention that PET did not do too well on the F-measure, 42.73, compared to the other participants—the top score was 50.79. The recall-precision graph for MUC-7 is shown in figure 5.12. It is curious to observe

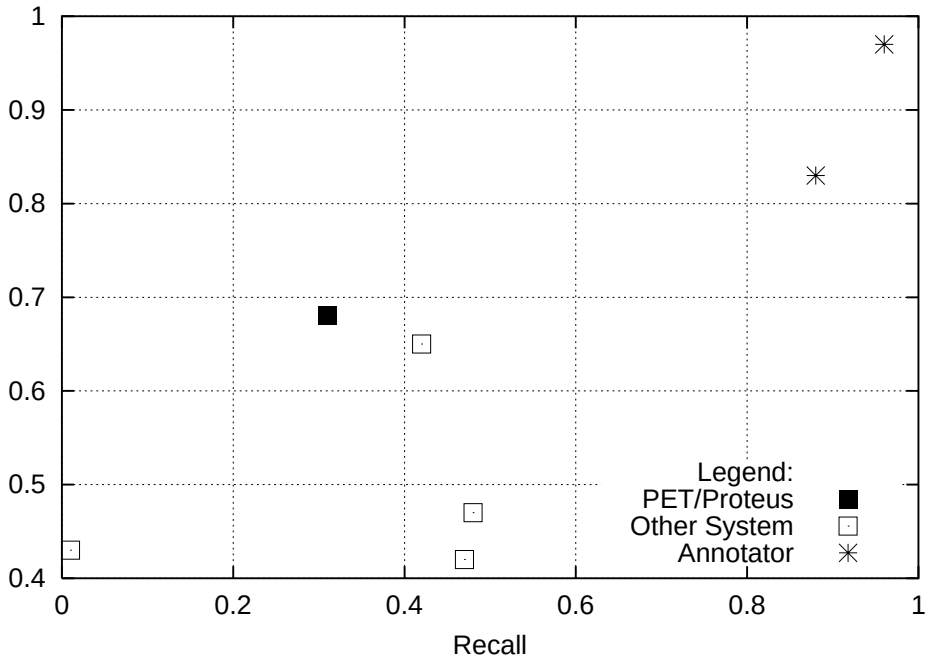

Figure 5.12: Performance on MUC-7 ST task

that PET/Proteus got the highest overall precision score, but a relatively low recall score. [5]

This experience suggested that although PET was good at helping the user transform examples into correct patterns, it did little to help her/him *find* good examples to begin with. The customization bottleneck was shifted from building patterns to finding them.

Thus came about the motivation for research into algorithms for pattern discovery, described in the next chapter.

[5]The graph also shows the performance of two human annotators who prepared the answer keys. One piece of evidence for the complexity of this task is the high degree of inter-annotator disagreement, and the poor performance of one of the annotators.

Chapter 6

ExDisco: Pattern Discovery

We now present the conceptual outline of ExDisco, our procedure for automatic acquisition of patterns. Chapter 7 elaborates our methodology in greater detail. The procedure is *unsupervised* in that it does not require the training corpus to be manually annotated with events of interest, nor a *pre-classified corpus* with relevance judgements, nor any feedback or intervention from the user. The procedure is based on an iterative process, with small improvements gained at each step.

6.1 Motivation

The central idea is to combine IR-style document selection with an *iterative relaxation* process. Similar techniques have been used elsewhere in NLP, and this idea was inspired in large part, if remotely, by the work of [29] on automatic alignment of sentences and words in a bilingual corpus for machine translation. The authors made two crucial and mutually informing observations:

- Sentences that are translations of each other are good indicators that words

they contain are translation pairs.

- Conversely, words that are translation pairs indicate that the sentences which contain them correspond to one another.

Thus there is a *duality* between the space of word correspondences and the space of sentence pairings, and the two spaces are inherently mutually connected.

In our setting, in the context of discovering good patterns for information extraction, we base our motivation on two fundamental assumptions or principles: the principle of *density* and the principle of *duality*.

The Principle of Density: *if* we are given a set of texts which has been partitioned into relevant vs. non-relevant subsets, *then* it stands to reason that those patterns which are much more strongly correlated with the relevant subset than with the non-relevant subset will generally be good patterns for the domain.

This observation is similar to one made in [45], where the author states:

> "The motivation [...] is that domain-specific expressions will appear substantially more often in relevant texts than irrelevant texts."

and where this principle was used successfully to extract events describing terrorist attacks. The second key observation is that there is a duality between the space of documents and space of patterns.

The Principle of Duality:

I. Documents that are relevant to the scenario are strong indicators of good patterns.

II. Conversely, good patterns are strong indicators of relevant documents.

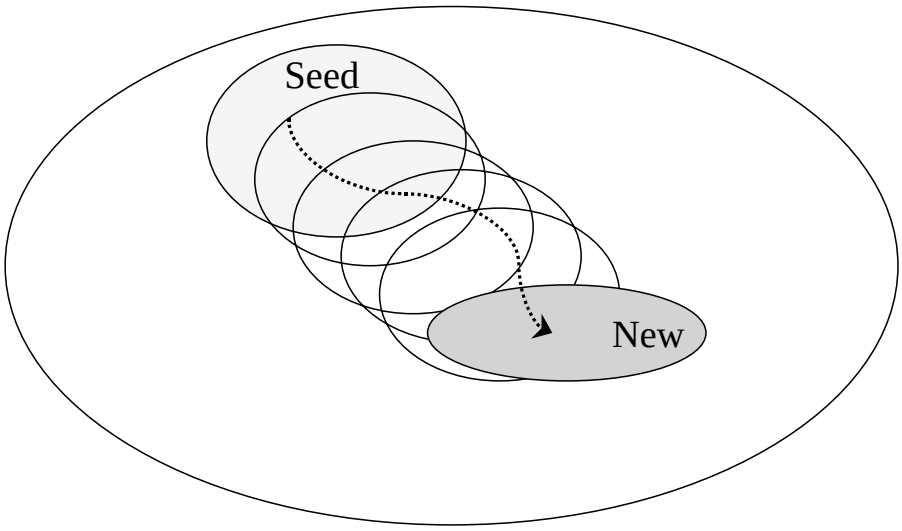

Figure 6.1: Intuition for Iterative Discovery

Part I of the duality principle is just a restatement of the density principle. Part II is the *converse* of part I, which is also intuitively true. The two together allow us to exploit the pattern which receive highest score to *expand* the set of documents believed to be relevant. The intuition that motivates the iterative procedure is diagrammed in figure 6.1. Starting with a seed set of patterns and documents, we proceed to identify more patterns and relevant documents in a bootstrapping chain.

6.2 Outline of ExDisco

0. **Given**:

   (a) a large corpus of *un-annotated and un-classified* documents;

   (b) a **trusted** set of scenario patterns, initially chosen *ad hoc* by the user—

the *seed*. As will be seen, the seed can be quite small—two or three patterns seem to suffice;

(c) an initial (possibly empty) set of concept classes.

1. **Partition**: The trusted patterns induce a binary split on the corpus: for any document, either zero or more than zero patterns will match it. Thus the universe of documents, U, is partitioned into the relevant sub-corpus, R, vs. the non-relevant sub-corpus, $\overline{R} = U - R$, with respect to the given pattern set. Actually, the documents are assigned weights between 0 and 1; documents matched by the seed receive the weight 1.

2. **Search for new candidate patterns**:

   (a) Automatically convert each sentence in the corpus into a set of candidate patterns.[1]

   (b) Working from the relevant documents, consider those candidate patterns, p, which meet the *density* criterion:

$$\frac{|H \cap R|}{|H|} \gg \frac{|R|}{|U|} \tag{6.1}$$

   where $H = H(p)$ is the set of documents where p hits. The key idea is to select those patterns whose distribution is strongly correlated with the relevant documents, i.e., those patterns that are distributed much more densely among the relevant documents than among the non-relevant ones.

[1]For each clause in the sentence we extract a tuple of its major roles: the head of the subject, the verb group, the object, object complement, as described in the next chapter. This tuple is considered to be a pattern for the present purposes of discovery. It constitutes a skeleton for the rich, syntactically transformed patterns which our system uses in the extraction phase.

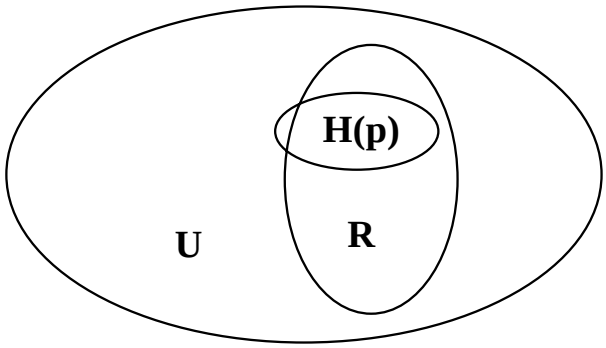

Figure 6.2: Density of Pattern Distribution

We select the best pattern, the one with the strongest correlation, and add it to the growing trusted set.

3. **Find new concepts** *(optional)*: Based on co-occurrence with the newly chosen pattern, extend the concept classes.

4. **User feedback** *(optional)*: Present the new candidate and classes to the user for review, retaining those relevant to the scenario.

5. **Repeat:** The new trusted pattern set induces a new partition on the corpus. With this pattern set, return to step 1. Repeat the procedure until no more patterns can be added.

6.3 Notes

A few points are important to observe in regard to this approach:

- The intuition behind formula 6.1 can be visualized as figure 6.2. The term

on the right side of the inequality is an estimate of the marginal probability of relevance; given no other information, this is about how often we would expect to find a relevant document in the corpus. The term on the left side is an estimate of the conditional probability of relevance; it says, given that the pattern p has matched a document, how likely is it to be relevant.

The term on the right is fixed for a given corpus, during a single iteration; therefore finding the best pattern amounts to maximizing the left side. This forms the basis for our scoring function in the next chapter, (see section 7.5).

- As presented so far, this process may result in very expansive search, potentially returning all patterns about a given topic, rather than only generalizing the *specific* relations sought by user. However, it seems an intuitively convincing point that if some structure (syntactic or semantic) is strongly correlated with the relevant sub-corpus, then this structure bears information relevant to the scenario. Thus, the discovery procedure may be used to suggest new entries for the *predicate* base.

- The same general approach may be adaptable to populate lower-level knowledge bases as well, i.e., the lexicon and the concept base. We will return to this point in a later section, in relation to the predicate base.

- The procedure may be supervised at the end of each iteration. We have considered a variant of the procedure which allows the user to check the results of the current iteration to improve the overall quality. We would expect that verification from the user would generally improve the overall quality of the results. However, we could not show substantial improvements with this vari-

ant in initial experiments. Results on this point remain inconclusive for now, and need to be researched further.

Chapter 7

Methodology

Before applying the discovery procedure, we subject the corpus to several pre-processing steps.

7.1 Pre-processing: Name Normalization

First we apply the Name Recognition module of Proteus, and replace each proper name with a special token describing its class, e.g. *C-Person*, *C-Company*, *C-Location*, etc. We also collapse together all numeric expressions such as percentages, currency values, dates, etc., using a single token to designate each of these categories. While factoring names and other out-of-dictionary items (OODs) out of the text makes the parser's job easier, it is *essential* to the discovery procedure—it allows us to maximize the leverage of commonality or redundancy in text for finding repeated patterns.

7.2 Pre-processing: Syntactic Analysis

After the OODs have been factored out, we run the corpus through a syntactic parser. We use a general-purpose dependency parser of English, based on the FDG formalism [52] and developed by the Research Unit for Multilingual Language Technology at the University of Helsinki, and Conexor Oy. The parser's output is used for reducing each clause or noun phrase to a tuple, consisting of several arguments.[1]

1. For clauses, the first argument is the subject, a "semantic" subject of a non-finite sentence or agent of the passive:

   - "**John** *retired*",
   - "John is *appointed* by **Company**",
   - "He succeeds **John** who *retired*",
   - "The board pushed **John** to *resign* his post."

2. The second argument is the verb. The parser finds the main verb of the clause, keeping only the base form of the head verb in a verb chain, removing tense information and auxiliaries. For subordinate clauses the parser attempts to reconstruct the missing semantic roles. E.g., the sentence

   - "*John Smith* **resigned**, **to join** Sony, Inc."

   would yield two clauses, with main verbs "resign" and "join".

3. The third argument is the object, certain object-like adverbs, subject of the passive or subject complement:

[1] Transformations of Conexor parser output were previously described in [57, 58]

- "**John** is *appointed* by Company",
- "John *is* the **president** of Company",
- "The board *reprimanded* **the chairman**, who resigned",
- "The **chairman** whom the board *reprimanded* resigned."

4. The fourth argument is a phrase which refers to the object or the subject. A typical example of this argument is an object complement, such as

   - "The company named *John Smith* **president**."

   This argument can also be filled with the so-called *copredicative* [41], in the parsing system [28]. A copredicative is a NP that refers to a subject or an object, though this distinction is typically difficult to resolve automatically. For example:

   - "She gave us our *coffee* **black**",
   - "*We* took a swim **naked**",
   - "*Bill Clinton* appointed *John Smith* as **president**".

   The ambiguity in last example shows why it is difficult to resolve the reference purely on syntactic grounds, so the the fourth argument may refer to either the subject or the object.

5. The fifth argument contains a locative or a temporal modifier. Because this is usually a prepositional phrase, the parser actually returns a pair: the preposition and the head of the modifier.

6. The last clausal argument is a prepositional phrase for which the parser could not determine the proper attachment. This is the so-called "high-attachment" modifier, attached at the clause level by default.

Each NP tuple contains four arguments:

1. the head noun of the NP,
2. the possessive pre-modifier—e.g., "**John Smith's** retirement",
3. a noun pre-modifier, e.g., "the **Challenger** disaster"
4. a post-modifier or apposition, "the nomination of **John Smith**".

The parser performs several levels of syntactic *normalization*. The purpose of normalization is to maximize commonality among the patterns occurring in the text.

One type of normalization is the transformation of clausal variants, such as the various passive and relative clauses, into a common form. This includes attempts by the parser to resolve relative pronouns, so that "Sam hired John", and "... John, who was hired by Sam" yield identical structures.

The second type of normalization is the reduction of inflected forms to corresponding *base* forms. This is done for the heads of verb groups and noun phrases, by stripping auxiliaries, tense, number, case (for personal pronouns), etc.

In the experiments reported in this thesis, we did not use all of the rich syntactic information returned by the parser. For now, we do not use the NP tuples at all. Each clausal tuple is further reduced to a *triple*; we keep only the subject, the verb and the object.[2] When the object is missing and the fourth argument is filled, it is

[2]Note that the subject or the object is *missing* if the parser is unable to resolve it.

moved into the third position in the final triple.

Because we are restricting ExDisco to a triple for each clause, this last transformation is intended to capture more information for verbs for which the parser routinely produces an empty object slot, but a non-empty fourth argument, such as "be", "remain", etc.[3]

The subsequent phases of ExDisco operate on the triples obtained as a result of this phase. Subsequently we will refer to the arguments of each triple as subject, verb, and *object*, although the last argument should more precisely be called *object-or-complement*.

7.3 Generalization and Concept Classes

The resulting tuples may not repeat in the corpus with sufficient frequency to obtain reliable statistics. For this reason, each tuple is further generalized into a set of pairs: i.e., a *verb–object* pair, a *subject–object* pair, etc. The phase of ExDisco which searches for patterns actually works on these pairs, or *generalized patterns*.

Once ExDisco has identified a *pair* as relevant to the scenario, it uses the values for the missing role to construct a new concept class. This is done by grouping together into one class the values for argument which is missing in the generalized triple. For example, suppose ExDisco decided that the best candidate

[3]In the results discussed in chapter 8, this transformation is incomplete; we moved the object complement to the third position only for the verbs *be* and *become*, since we initially assumed these verbs to be special. Subsequent analysis revealed a large class of verbs for which the parser analyzes a subject/object complement with a missing object, e.g., *stay*, *remain*, *consider*. We intend to conduct experiments with a uniform transformation independent of the verb.

(generalized pair) is "*company*–*–*person*", a SVO triple with the verb missing. It would then form a concept class for the verbs which occur with this subject–object pair: "*company* {hire/fire/expel...} *person*".

Note that ExDisco does not admit *all* values of the missing role as members of the new concept class. Such indiscriminate inclusion would yield completely irrelevant triples, such as "company–sue–person" or "company–congratulate–person". Each value for the missing role gives rise to a full, non-generalized triple—we call this a *primary* triple. ExDisco chooses only values corresponding to primary triples that themselves have high correlation scores, i.e. those that are more densely distributed among the relevant documents than overall. The correlation score is computed by the same scoring scheme as detailed below in section 7.5, using a fixed cutoff.

Several other groups have explored induction of semantic classes through analysis of syntactic co-occurrence, [46, 43, 12, 25], though in our case, the contexts are limited to selected syntactic constructs which are relevant to the scenario.

7.4 Indexing

From the primary triples consisting of the main three clausal arguments, we build an inverted index back into the parsed documents. Each clause annotation produced by the parser has (possibly) a subject, a verb, and (possibly) an object. The triple consisting of the syntactic heads of the group serves as the key for the indexing.

A second index maps each generalized triple into a set of corresponding primary triples.

7.5 Pattern Ranking

ExDisco ranks all patterns — precisely, the generalized triples — according the strength of their correlation with the relevant sub-corpus. Let us define the *correlation score* function, $Score(p)$ as[4]:

$$Score(p) = \frac{|H \cap R|}{|H|} \cdot \log |H \cap R| \tag{7.1}$$

where R denotes the relevant subset of documents, and $H = H(p)$ the documents matching p, as above. As observed in section 6.3, the first term accounts for the conditional probability of relevance given the pattern p, and the second is a measure of support for p among the relevant documents.

We impose two further *support criteria*: we distrust such frequent patterns where $|H \cap U| > \alpha|U|$ as uninformative, and rare patterns for which $|H \cap R| < \beta$ as noise.[5] At the end of each iteration, the system selects the pattern with the highest correlation score, and adds it to the trusted set. The documents which the winning pattern matched are added to the relevant set. The search for patterns is then restarted.

7.6 Re-computation of Document Relevance

The above description is simplified in several important respects.

ExDisco assigns relevance scores to documents on a scale between 0 and 1. The seed patterns are accepted as ground truth; thus the documents they match have relevance 1. On subsequent iterations, the newly accepted patterns are not trusted

[4]Similarly to that used in [45]

[5]We used $\alpha = 0.1$ and $\beta = 2$.

as absolutely. On iteration number $i+1$, each pattern p is assigned a precision measure, based on the relevance of the documents it matches:

$$Prec^{i+1}(p) = \frac{1}{|H(p)|} \cdot \sum_{d \in H(p)} Rel^i(d) \tag{7.2}$$

where $Rel^i(d)$ is the relevance of the document from the previous iteration, and $H(p)$ is the set of documents where p matched. In general, if K is a classifier consisting of a *set* of patterns, we can define $H(K)$ as the set of documents where *all* of patterns $p \in K$ match, and the "cumulative" precision of K as

$$Prec^{i+1}(K) = \frac{1}{|H(K)|} \cdot \sum_{d \in H(K)} Rel^i(d) \tag{7.3}$$

Once the new winning pattern is accepted, the relevance scores of the documents are re-adjusted as follows. For each document d which is matched by some (non-empty) subset of the currently accepted patterns, we can view that subset of patterns as a classifier $K_d = \{p_j\}$. These patterns determine the new relevance score of the document as

$$Rel^{i+1}(d) = \max\left(Rel^i(d), Prec^{i+1}(K_d)\right) \tag{7.4}$$

This ensures that the relevance score grows monotonically, and only when there is sufficient positive evidence, as the patterns in effect vote "conjunctively" on the documents. The experimental results which follow use this measure. Alternatively, we can adopt a more relaxed, "disjunctive" voting scheme, and let the patterns in K_d vote on the new relevance of d, as in:

$$Rel^{i+1}(d) = 1 - \prod_{p \in K_d} \left(1 - Prec^{i+1}(p)\right) \tag{7.5}$$

or add weights to the voting, to account for variation in *support* of the patterns,

$$Rel(d) = 1 - \sqrt[\overline{w}]{\prod_{p \in K_d} \Big(1 - Prec(p)\Big)^{w_p}} \tag{7.6}$$

where the weights w_p are defined using the relevance of the documents, as the total support for pattern p:

$$w_p = \log \sum_{d \in H(p)} Rel(d)$$

and $\overline{w}$ is the largest weight. Thus in formula 7.1 above, $|H \cap R|$ is not simply the count of the relevant documents, but is rather their *cumulative relevance*, i.e., $\sum Rel(d)$. The recursive pairs of formulas, (7.2) and (7.5), or (7.3) and (7.4), capture the mutual duality between patterns and documents[6]. This re-computation and growing of precision and relevance scores is at the heart of the procedure.

7.7 Discovering Actions

We observed earlier that patterns consist of two parts: the triggering expression and an action or a mapping from the syntactic to the semantic roles. The above sections have focused exclusively on the discovery of triggers. We can begin to address the problem of discovering actions in a similar fashion by bootstrapping. The discovery procedure for actions would run as follows.

For the seed pattern triggers, the user specifies the mapping explicitly. Once we have acquired a set of good triggers, we can then study the occurrences of the triggers in actual text. The seed triggers induce a set of *instances*; an instance is a tuple of pattern arguments whose relation is known. Given enough data, we would

[6]We did not observe a significant difference in performance between the two formulas 7.4 and 7.6 in our experiments; the results which follow use 7.6.

expect that important events are described by several instances, and in different ways.[7]

At this point we could exploit the duality between instances on one hand and triggers on the other: if a given trigger matches an instance whose elements stand in a known relation to each other, we can infer the relation induced by this trigger. On the other hand, each new trigger inducing a given relation will produce more instances of that relation. This calls for a bootstrapping algorithm to fill out the relations induced by the triggers discovered by our procedure above; this is not implemented currently and remains to be investigated.

[7]This observation is closely related to one on which the DIPRE approach is predicated, see chapter 2 on prior work.

Chapter 8

Experimental Results

Establishing an objective measure of goodness of patterns discovered by ExDisco is not a straightforward matter. Evaluation is complicated by several factors.

First, the discovered patterns are in a sense incomplete: they cannot be used for information extraction directly. The patterns carry only *trigger* information, and lack action information, i.e., which events a pattern induces and how the pattern arguments are used to fill event slots.

Second, to conduct a full, MUC-style IE evaluation, the patterns must first be properly incorporated into the Proteus knowledge bases. As ExDisco stands alone, incorporating the patterns into Proteus requires additional manual effort.

Third, even if we were to incorporate the discovered patterns into some form of our IE system, in order to have an objective evaluation, we would need a set of "correct answers" against which to compare the system's results. Such correct answers are very expensive to produce, as they require human annotation, which takes great amounts of time.

The "correct answers"—which can be used for an objective evaluation—can

obtain at different level of detail. At the level of greatest detailed are the standard MUC ST answer templates. In this form, the answer specifies for each document which events are to be extracted and how the slots in each event are to be filled. Alternatively, we may have answers only at the level of *relevant documents*. This corresponds exactly to relevance judgements which constitute the "correct answer" in IR evaluations, like the Text Retrieval Conferences (TREC) evaluations, cf., e.g., [50].

With this in view, in this chapter we propose three different measures of performance:

- **Qualitative evaluation**: We begin by manually inspecting the extracted patterns, and attempt judge their quality without resorting to quantitative measures. Thus, since it rests completely on the reviewer's intuition, this will not be an altogether objective method. However, it will provide a fairly good sense of how well the patterns discovered by ExDisco match the scenario.

- **Text filtering**: If we have a set of documents with relevance judgements, we can obtain an effective measure by noting that, in addition to growing the pattern set, ExDisco also grows the relevance rankings of documents. The quality of retrieved documents can be evaluated directly, without human intervention. This view on the discovery procedure is closely related to the MUC "text-filtering" task, in which the systems are judged only at the level of *documents*, as in IR, rather than at the level of events and slots.

- **Event extraction**: If we have a set of test documents with answer keys—correctly filled templates—for the scenario, we can evaluate ExDisco by

manually incorporating the discovered patterns into the knowledge bases and running a full MUC-style extraction with scoring.

We present the different types of evaluations in turn.

8.1 Qualitative Evaluation

We tested ExDisco on several scenarios in the domains of business news and general news. For business news, we chose a corpus of 9,224 articles from the Wall Street Journal. The articles were drawn from a two month period in 1992 and one month in 1994. The parsed articles yielded a total of 440,000 clausal tuples, of which 215,000 were distinct.

For general news, we used a corpus of 14,000 articles from Associated press, covering the first three months of 1989.

8.1.1 Management Succession

We tested ExDisco most extensively under the Management Succession scenario, since for this scenario we have pre-existing extraction systems and high-quality Training and Test data (cf. section 2.1). We start with the seed in table 8.1.

Subject	*Verb*	*Direct Object*
C-Company	V-Appoint	C-Person
C-Person	V-Resign	—

Table 8.1: Seed pattern set for "Management Succession"

The tokens *C-Company* and *C-Person* denote semantic classes containing named

entities of the corresponding types. *V-Appoint* denotes the list of verbs { *appoint, elect, promote, name, nominate* }, *V-Resign* = { *resign, depart, quit* }.

The following is a list of some of the patterns discovered by ExDisco over the first 80 iterations. The list shows the SVO triples, together with additional (bracketed) constituents, which are outside of the central SVO triple, and are included here for clarity. The list includes several examples of "bad" elements for demonstration purposes—these are underlined.

1. person-succeed-person
2. person-{be,become}-president
3. person-become-{chairman,officer,president,executive}
4. person-retire-
5. person-serve-{company,board,sentence}
6. person-{run,leave,join,say,own,head,found,
   ... start,remain,rejoin}-company
7. person-assume-{post,title,responsibility,control,duty}
8. person-replace-person
9. person-step-[down]-[as-officer]
10. person-{relinquish,assume,hold,accept,retain,
    ... take,resign,leave}-post
11. company-name-{chairman,president,successor}

12. *-hire-person

13. person-run-{company,operation,campaign,organization,business}

14. person-hold-{position,title,post,stake,share,job,meeting}

15. person-{hold,retain,resign,fill}-position

16. person-take-{control,helm,care,action,office,retirement,
    ... job,post,duty,responsibility}

Evidently, ExDisco discovers useful patterns. It is interesting, if sobering, to observe that the pattern person-serve-sentence turns out to be strongly correlated with this domain...

A simple measure of performance is comparing the patterns found by the procedure with those in the extraction engine which was manually constructed for the same task. Our MUC-6 system used approximately 75 clause level patterns, with 30 distinct verbal heads. In one conservative experiment, we observed that the discovery procedure found 17 of these verbs, or 57%. However, it also found many clausal patterns which were missing from the manually constructed system, and which are strongly relevant to the scenario, e.g.,:

company-bring-person-[in]-[as+officer]

person-{come,return}-[to+company]-[as+officer]

person-rejoin-company-[as+officer]

person-tap-person-[to-be-officer]

person-{continue,remain,stay}-[as+officer]

person-pursue-interest

At the risk of setting off a philosophical debate over what is or is not relevant to a scenario, we note that the first four of these are directly essential to the scenario, as they convey a change of post.

The next pattern contains "staying" verbs. Although Proteus does not have such patterns, these patterns are actually also needed in certain special cases, which are covered in the MUC-6 ST task specifications.

Namely, in a case when a person occupying some explicitly stated post, P_{prev}, assumes a new post, P_{new} in another company, the fill rules automatically assume an implicit transition in which the person left P_{prev}, and mandate that the system generate a corresponding "departure" event. However, when explicit information to the contrary is available, e.g., as conveyed by the "staying" patterns, the system has sufficient information *not* to generate a "departure" event.

The pattern "`person-pursue-interest`" is the most curious of those found by ExDisco. Surprisingly, it too is useful, even in the strictest MUC-6 sense, cf. [39]. According to the task specifications, for each succession event, the system must fill a slot called *"other-organization"*. This slot contains the name of the organization from which the person came to the new post or to which s/he went from the old post. Whenever such information is available in the text it must be used to fill the slot. With the aid of ExDisco, we learned that this particular pattern is consistently used in text to indicate that the person left to pursue *other, typically undisclosed* interests. Having this clue would relieve the system from seeking other information to fill the "other-organization" slot.

We should note that in preparation for MUC-6, the designers of Proteus studied not only the 100-document Training corpus, but also a large number of additional

documents retrieved with IR queries. These new patterns, found by ExDisco did not come up.

We should also note in this context that at the time of the MUC-6 competition Proteus achieved the highest F-score of all participants.

8.1.2 Mergers and Acquisitions

For this scenario, we used the seed in table 8.2.

Subject	*Verb*	*Direct Object*
*	V-Buy	C-Company
C-Company	merge	*

Table 8.2: Seed pattern set for "Mergers and Acquisitions"

The class *V-Buy* contains two verbs: { *buy, purchase* }. ExDisco found the following patterns (among others):

```
*-complete-purchase
company-express-interest
company-seek-partner
company-acquire-{business,company,stake,interest}
company-{acquire,have,own,take[over],pay,drop,sell}-company
company-{have,value,acquire}-asset
{company,company-unit,bank}-have-asset
*-expect-transaction
acquisition-be-step
company-own-{company,station,stake,store,business}
```

```
company-{hold,buy,own,raise,pay,acquire,sell,
...          boost,take,swap,retain}-stake
company-hold-{stake,percent,talk,interest,share,position}
company-buy-{stake,share}
company-have-{sale,asset,earning,revenue}
company-{issue,have,repurchase,own,sell,buy,hold}-share
company-{report,have,post,expect,record}-loss
company-report-{loss,earning,income,profit,increase,sale}
company-{become,hire,sell,control,invest,
...          compete,make}-company
```

8.1.3 Corporate Lawsuits

For this scenario, we used the seed in table 8.3.

Subject	*Verb*	*Direct Object*
*	V-Sue	C-Organization
*	bring	suit

Table 8.3: Seed pattern set for "Corporate Lawsuits"

The class *V-Sue* contains two verbs: { *sue, litigate* }. This small seed induces a large number of relevant patterns:

```
person-hear-case
plaintiff-seek-damage
group-charge-[that ...  ]
```

```
*-face-suit
company-appeal-
we-marshal-argument
they-recoup-loss
they-alert-regulator
company-abet-violation
lawsuit-allege-[that ... ]
suit-seek-[that ... ]
assets-be-frozen
{person,government-org}-file-suit
{person,court}-reject-argument
person-rule-[that ...]
company-deny-{allegation,charge,wrongdoing}
company-settle-charge
court-rule-[that ... ]
```

8.1.4 Natural Disasters

For this scenario, we used the seed in table 8.4.

Subject	*Verb*	*Direct Object*
C-Disaster	cause	*
C-Disaster	V-damage	C-Urban

Table 8.4: Seed pattern set for "Natural Disasters"

The concept classes used here are: *C-Disaster* = { *earthquake, tornado, flood,*

hurricane, landslide, snowstorm, avalanche }, *V-Damage* = { *hit, destroy, ravage, damage* } , *C-Urban* = { *street, bridge, house, home, —* }.

This seed induces the following relevant patterns:

```
quake-measured-(number)
quake-was-felt
earthquake-shook-area
earthquake-was-centered
quake-struck-
quake-was-considered
quake-occurred-
quake-knocked-[out]-power
storm-knocked-[out]-power
damage-was-reported
aftershock-injured-
aftershock-killed-people
quake-registered-
it-caused-damage
quake-{killed,injured}-people
```

8.2 Text Filtering

As new patterns are discovered, documents that contain these patterns gain in relevance weight. In this evaluation, we consider the quality of documents which ExDisco deems relevant. For the purposes of judging precision and recall, a docu-

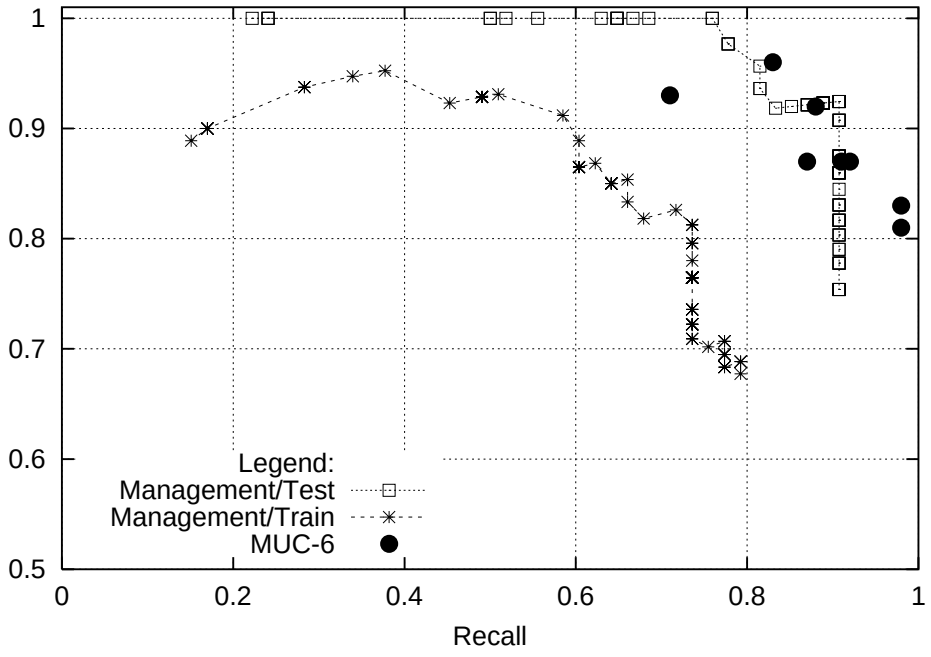

Figure 8.1: Management Succession

ment is counted as relevant if its internal relevance score reaches a pre-determined threshold θ. In this way we can view ExDisco as an IR engine and evaluate it as such. We compared ExDisco's behavior under different thresholds, and empirically, $\theta = 0.5$ appears to give the best results.

This gives us a binary relevance judgement for each document in the collection, and we compute recall and precision according to the IR-style formulas 4.1 and 4.2.

For the MUC-6 task, we tested ExDisco against the two MUC-6 corpora, using only the relevance information from the answer keys. Both of these sets were included among the 10,000 articles on which ExDisco was trained. These judgements constitute the hidden truth which was used only for evaluation, *not* visible to ExDisco.

Figure 8.1 shows recall plotted against precision on the two corpora, over 100 iterations, starting with the seed patterns in section 8.1. We observe a similar trend in both curves, where recall grows steadily with reasonable precision until at some point bad patterns invade the system and improvement stops. The difference between the two curves is about 10 points of recall. We can only speculate about the possible cause for this, since the Test corpus remains unrevealed. It is likely due to the fact that some patterns are more easily learned by ExDisco than others, and these patterns happen to occur in larger proportion in the documents in the Test corpus.

It is interesting to compare ExDisco's text filtering results on the *MUC-6 Test* corpus with those of the other MUC-6 participants, shown anonymously in the plot. ExDisco attains values well within the range of the MUC participants, all of which were either heavily-supervised or manually coded systems. It is important to bear in mind that ExDisco had no benefit of training material, or any information beyond the seed pattern set.

Figure 8.2 shows the performance of text filtering on the Acquisition task, again, given the seed in table 8.2. ExDisco was trained on the same WSJ corpus, and tested against a set of 200 documents. We retrieved this set using keyword-based IR search, and judged their relevance by hand.

8.3 Event Extraction

We now describe how we evaluated ExDisco on the slot-filling task. We *manually* incorporated the discovered patterns into the Proteus knowledge bases and ran a full MUC-style extraction against the original MUC-6 Training and Test corpora,

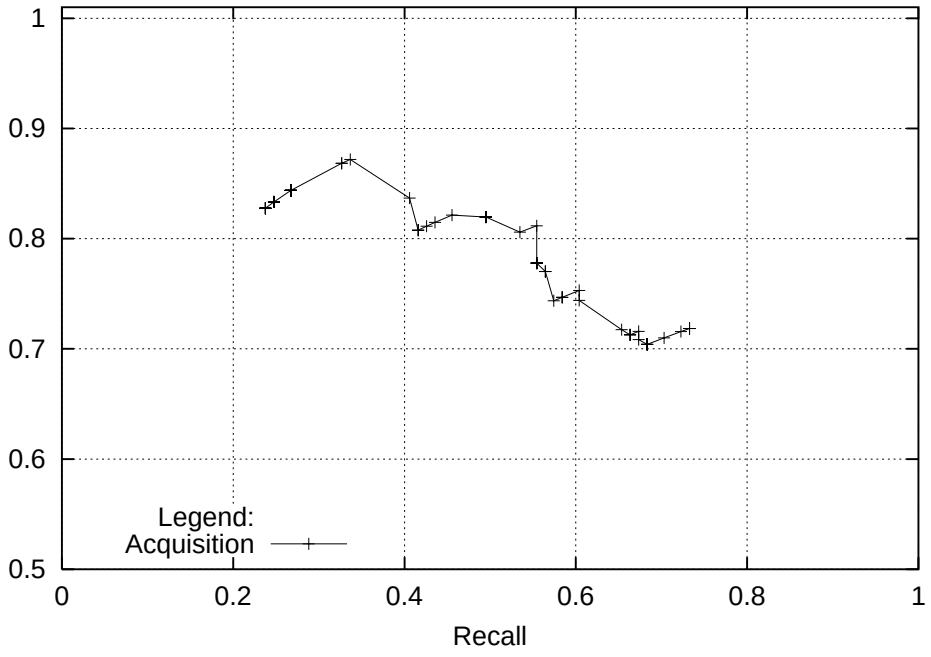

Figure 8.2: Mergers/Acquisitions

each containing 100 Wall Street Journal articles. We then took the results produced by Proteus and applied full MUC scoring to them. The recall and precision on the MUC-6 corpora are shown in table 8.3. We started with a version of *Proteus* which was used in MUC-6 in 1995 and had undergone continual improvements since the MUC evaluation. We removed all scenario-specific clause and nominalization patterns.[1] We then reviewed the patterns that were generated by ExDisco, and discarded those which were not relevant to the task, and those which did not correspond directly to a predicate already implemented for this task. This is worth

[1]There are also a few *noun phrase* patterns which can give rise to scenario events. For example, "Mr Smith, former president of IBM,...", may induce an event record where Fred Smith left IBM. These patterns were left inside *Proteus* for all runs, and they contribute to the relatively high baseline scores obtained using just the seed event patterns.

	Training			Test		
Pattern Base	*Recall*	*Precision*	*F*	*Recall*	*Precision*	*F*
Seed	38	83	52.60	27	74	39.58
ExDisco	62	80	69.94	52	72	60.16
Union	69	79	73.50	57	73	63.56
Manual-MUC	54	71	61.93	47	70	56.40
Manual-NOW	69	79	73.91	56	75	64.04

Table 8.5: Performance of slot filling on "Management Succession"

mentioning, since ExDisco discovered several relevant patterns which could not be accommodated in Proteus in its current state. These were the "staying-on-a-job" patterns, discussed in section 8.1.1, for instance "Person remained president".

To use the "staying" patterns we would have to make a set of additional inference rules. In cases where the system infers a departure from some combination of events, these rules should check for the presence of a "staying" event and override the implicit departure. We have not yet added such "suppression" rules to Proteus to accommodate the "staying" patterns.

The remaining, "good" patterns were augmented with actions—i.e., information about the corresponding predicate, and the mapping from the pattern's constituents to predicate arguments. As all clause-level patterns in Proteus, the resulting patterns are automatically generalized to handle syntactic variants such as passive, relative clause, etc. The resulting pattern bases were applied to the MUC-6 Formal Training and Formal Test corpora, and the output evaluated with the MUC scorer.

The *Seed* pattern base consists of just the initial pattern set, given in table 8.1.

The seed patterns alone give result in a recall of 38% and precision of 83%, for a total F-measure of 52.60. This serves as the baseline for the Training corpus.

We then added in the patterns which the system discovered automatically, producing the pattern base called ExDisco. That gave us a 24 point rise on recall and a loss of 3 points on precision, boosting the F-measure by over 17F points to almost 70F. For comparison, our official result on this corpus was just under 62F—this corresponds to the pattern base labeled *Manual-MUC*. This pattern base was manually prepared for the MUC-6 Training corpus over the course of 1 month of full-time work by at least one computational linguist (1.5 linguists, actually), who studied the 100-document Training corpus in depth over that time. The last row, *Manual-Now*, shows the current performance of the Proteus system, which is just under 74F points. The base called *Union* contains the union of ExDisco and *Manual-Now*.

Similarly, we observe a gain of over 20F points on the Formal Test corpus, when going from the seed pattern base to ExDisco. The resulting score of over 60F is again higher than the official Proteus MUC-6 score.

We find these results encouraging: Proteus performs better with the patterns discovered by ExDisco than it did after one month of manual tuning and development; in fact, this performance is close to current levels, which are the result of substantial additional development.

Further notes concerning these results follow in the discussion in the next section.

8.4 Caveats: Management Succession

The results on the slot filling task for the Management Succession task, reported above should be interpreted with several caveats in mind.

1. The overall performance of the IE system depends on many factors, and although the event patterns in some sense play a principal role, the remaining components of the underlying engine impact performance as well. Those components, including name recognition, syntactic analysis, anaphora resolution, and the inference engine, have been improved since the formal MUC-6 evaluation in 1995.

   In the formal evaluations, reported in table 8.3, the first three bases—*Seed*, ExDisco and *Union*—all had the benefit of running on top of the improved Proteus. Therefore some of the gain over the MUC formal evaluation score—the *Manual-MUC* base —may be attributable to these improvements. However, if we consider the *relative* improvements among the first three bases and *Manual-NOW*, these bases are commensurate in this regards.

2. As we noted above, before including the discovered patterns in Proteus, we reviewed the patterns, selected the good ones and augmented them by adding the appropriate actions to the triggers. Since this work was done manually, the overall, end-to-end procedure—i.e., discovery plus customization—is not entirely automatic.

   However, the review and augmentation process took little time, compared to the amount of time required for manual corpus analysis and development of the pattern base.

3. As we mentioned, the pattern base ExDisco in table 8.3 does *not* include the "staying-on-a-job" patterns discovered in section 8.1.1. Thus, in a way, ExDisco is not getting full credit for its discoveries in table 8.3. On the other hand, neither do the *manual* bases in table 8.3 get the benefit of such suppression, and it is difficult to speculate in retrospect how well the human developers would have done had they provided such a mechanism in Proteus from the outset.

Chapter 9

Discussion

We now turn to an analysis of the results presented in the preceding chapter, their relation to prior work, and problems that remain to be researched.

9.1 Utility Analysis

The evaluations in the preceding chapter confirm the effectiveness of ExDisco as a tool for finding high-quality patterns. We now turn to the question of its utility: is all this work necessary? How can we confirm whether it is necessary?

Consider the set of patterns Q, the final product of the discovery process for a given task, say, again, "Management Succession". Let us view Q as the "correct solution" to the problem of finding good patterns to capture the scenario. Let $N = |Q|$. This is diagrammed on the right side of figure 9.1.

Now consider the list of patterns which ExDisco evaluates on its very first iteration. Each pattern has some correlation score, according to the formula 7.1. Suppose we were to take the patterns reviewed on the first iteration and sort them

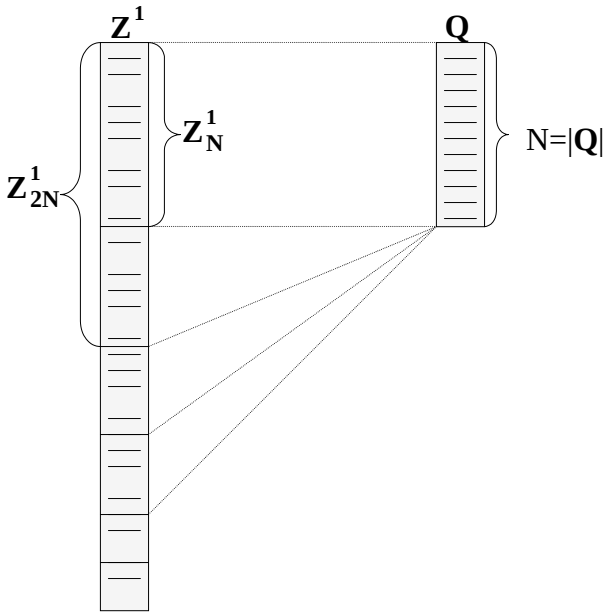

Figure 9.1: Patterns from First and Last Iteration

by correlation score. Let Z^1 be this sorted list (on the first iteration). Now we read off the first N patterns in Z^1, and let Z^1_N be this list of the N best patterns. How far off is Z^1_N from the correct solution Q? Clearly, if Z^1_N were already very close to Q, the usefulness of ExDisco is called into question. Suppose we include also the next N patterns in the list, Z^1_{2N} how much closer to the solution would that bring us?

The answer is shown in figure 9.2. We can view this as a retrieval problem: for each $j = 1, 2, ...$, compute recall and precision of the first $j * N$ patterns in Z^1 with respect to the "truth" given by the patterns in Q. The plot shows how recall and precision vary with j. This test was conducted on the "Management Succession" MUC-6 Training Corpus, over 65 iterations. Note that typically $|Z^i| \gg N$; in this

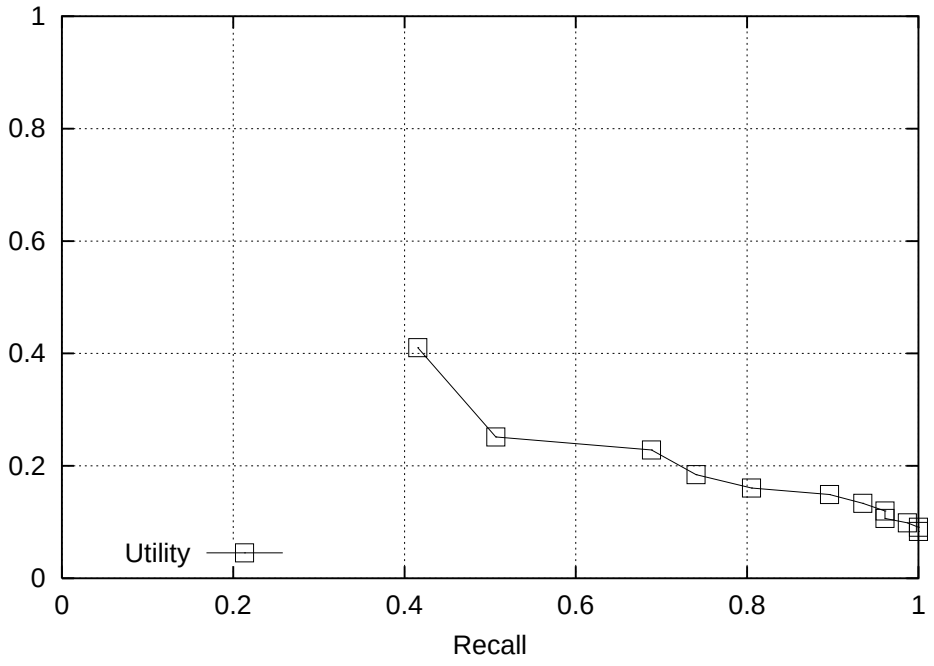

Figure 9.2: Utility of Z^1 against Q

test N was 65, and $|Z^1|$ was just under 1000.

If we use only the first N patterns of Z^1, we recover about 40% of the patterns in Q. If we also look through the next N patterns, we reach half of Q, and .22 of all the patterns we have to consider, or one in every four or five, appear in the final set. It is clear that Z^1 alone is not very helpful at finding good patterns—the cost of searching through the list for Q is large.

The situation is actually somewhat more complicated. We inspected Z^1_N thoroughly, paying particular attention to the relationship between Z^1_N and Q. We found that these patterns fall about evenly into three groups:

A. 27 patterns, or 41%, appeared in both.

B. 18 more patterns, or another 27%, of Z^1_N seemed good, but did not appear in

Q.

C. the remaining 20 patterns of Z^1_N were not in Q and were clearly irrelevant.

If we treat Q as the "gold standard", as we agreed at the outset of this section, then judging Z^1_N relative to Q leaves the precision and recall at 41%. However, this assumption may not be the most fair way to assess group B. If we assume a more liberal stance with respect to B, we could say that Z^1_N achieves $\frac{|A \cup B|}{N} = 0.69$ precision, but this would be *irrespective* of Q.

It is difficult to arrive at a strict judgement of set inclusion in this context, because we cannot claim that the patterns in group B were strictly *missing* from Q. In reality, all but one of the verbs in group B—`*-select-person`, were *subsumed* by other patterns in Q. This happens because the patterns under discussion here are *under-specified*—they are generalized patterns with one missing argument.

If we were to accept this subsumption argument, we would be moved to accept the stricter judgement of 41%. However, then one could counter-argue that patterns in Z^1_N subsume (at least partly) patterns in Q, and so Z^1_N merits a higher recall score with respect to Q.

If we abstain from the "gold standard" assumption, and judge Q on its own merit, we should observe that of the 65 patterns in Q, four or five were of questionable relevance, see section 9.2.1. This yields a minimum precision between 92% and 94% for ExDisco. By any measure, this still compares favorably with the figures for Z^1_N, which would lie somewhere between 41% and 69%.

Figure 9.3 shows how the situation changes on successive iterations. The lines plotted, starting from the bottom, are Z^i for $i = 1, 10, 20, ...$, every 10th iteration. The graph shows how the precision of Z^i_{jN} improves on each iteration.

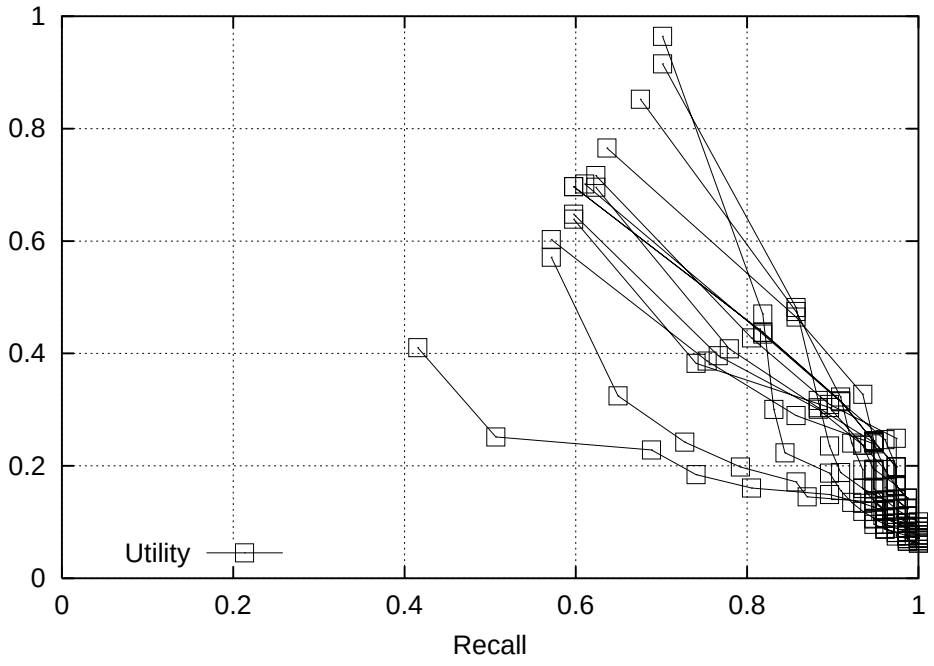

Figure 9.3: Utility across 80 iterations

This confirms the utility of ExDisco.

9.2 Analysis: Management Succession

We been analyzing the performance of ExDisco in its current state. We are at the early stages of understanding how best to tune these techniques, and the analysis suggests a number of areas that are in need of refinement.

As is common in NLP, the errors of ExDisco can be divided into two groups: errors of *commission* and errors of *omission*. The former degrade the precision scores, and the latter the recall.

9.2.1 Errors of Commission

In most cases, the patterns the procedure discovered were basically good, but occasionally included some irrelevant elements in the concept class for the expanded wild-card role. There were five potentially questionable patterns:

Bad_1 = `person-have-`*HaveObject*

Bad_2 = {`person,company`}-{`say,report`}-

where the class *HaveObject* is { *responsibility, plan, trouble, problem, experience, idea* }. The pattern Bad_1 was acquired quite late—iteration 61—and actually subsumes one potentially relevant pattern: `person-have-responsibility`, though the remaining objects in *HaveObject* are clearly non-relevant.

As for the patterns in Bad_2, it is unclear what modifications to the procedure might remove this type of error in principle. These patterns are weakly correlated with the scenario, since companies and their spokespersons usually announce major changes in corporate structure or operations, and management succession happens to be one instance of such major change. Thus, although we can expect these particular patterns to have wide support inside the relevant document set, we should as well expect wide support outside. With our current scoring function after several dozen iterations these weak-precision patterns reach the top of the ranked list.

The real difficulty in dealing with this problem is that we *don't* want to start modifying the scoring functions only to weed out this phenomenon, because often such weakly correlated patterns can be essential to the scenario. An example of this are the patterns which capture the number of casualties in the "Natural Disaster" scenario. Here, the fact that people die in *un-natural* disasters as well—giving wide support outside of the scenario—should not disqualify this valuable pattern's

candidacy. The same holds for patterns stating amount of damage incurred in natural disasters, etc.

We should note that errors of commission are in principle less troublesome in our setting, since they are easy to catch—by supervising the procedure. Errors of the second kind are much more problematic in this regard, and in fact they are what motivated the entire investigation into un-supervised learning techniques. We turn to these next.

9.2.2 Errors of Omission

As figure 8.1 shows, ExDisco cannot get beyond 80% recall on the MUC-6 Training Corpus without suffering a precipitous drop in precision. A study of the missing documents revealed, as expected, that several documents were not discovered because of missing patterns.

These patterns in turn were missing due to several kinds of errors. After about 70 iterations, there were twelve relevant documents still missing.

- Four of these documents contained good discovered patterns and had non-zero relevance scores, between .34 and .41. This fell below the threshold $\theta = 0.5$, which we chose *ad hoc*, cf. section 8.2. One of the documents contained the "weak" pattern 3 times, but ExDisco currently assigns no additional credit for multiple hits within a document.

- Five documents contained somewhat complex syntax, such as appositions and verbal conjunctions, which the parser could not resolve. For these no reasonable analysis was obtained and ExDisco had no hope of recovering the patterns.

- One document exhibited completely egregious behavior: it consisted of a single incomplete but long sentence, the event was partially stated in the body of the document, while the main verb was stranded in the *headline*.

- At least three documents contained truly missing patterns which could be recovered by possible extensions to ExDisco.

As shown in the lists of discovered patterns in section 8.1 on qualitative evaluation, on each iteration, ExDisco finds rich *classes* of related terms. These sets of terms are captured by the wild-card arguments of the generalized patterns. Currently, the procedure does not make use of the discovered classes *across iterations*. This area needs improvement. It would correspond to step 3 in the outlined procedure, section 6.2, where the classes are adjusted based on new relevant patterns.

More importantly, rather than merely growing the classes, we should dovetail the discovery of patterns and of concepts classes, i.e., to use the richer concept classes to boost the coverage of the patterns on subsequent iterations.

There is an obvious need for some form of generalization of patterns, and *merging* of patterns and concept classes to enrich the set of discovered patterns.

Preliminary analysis revealed several relevant documents which were not picked up because of missing patterns. Two such missing patterns were

P_m = `officer-take-retirement`, and

P_n = `executive-resign-post`.

Consider the data from the preceding chapter, section 8.1.1. Among the patterns discovered by ExDisco there were related patterns:

P_2 = person-{be,become}-president

P_3 = person-become-{chairman,officer,president,executive}

P_{11} = company-name-{chairman,president,successor}

Recall that the seed contains the pattern P_s = company-name-person. These and similar data would lead us to propose several types of mergers:

1. We should consider merging P_{11} with the seed pattern P_s, to extend the concept class to C = {chairman,president,successor,person}. This is intuitively justified because all terms in this class function as the object of the same subject-verb pair. We should be able to measure the strength of this association across different subject-verb pairs.

2. Another extension for the class C, to include officer, is supported by the evidence from the pattern P_3, and the overlap between the classes in these patterns. Again, the strength of the overlap can be measured across patterns.

3. The patterns P_2 and P_3 overlap in a complete clausal triple:

   person-become-president

   is common to both. This suggests merging the two patterns to generalize both the verb and object positions simultaneously:

   person-{be,become}-{chairman,officer,president,executive}

Another potentially fruitful method for extension of concept classes derives from the two central observations in [59]: namely, the founding principles that posit "one sense per discourse" and "one sense per collocation".

Suppose we find a certain subject-verb pair occurring with a particular object, `s-v-o`, among the accepted patterns. Suppose also that in one of the documents where `s-v-o` occurs, we find the tuple `s-v-o'`, with a different object. If we subscribe to Yarowsky's second principle, we can assume that the given *collocation*, the subject-verb pair `(s,v)`, is used in the same *sense* in the discourse. This should compel us to merge the two objects into a single class of related terms, `s-v-{o,o'}`.

This merging principle above assumed that `s-v-o` and `s-v-o'` occur in the same relevant document. It might be acceptable to relax the condition in one of two ways. One relaxation would be inspired by viewing the entire set of relevant documents as a single, coherent *discourse*. If we subscribe to this view, we could invoke Yarowsky's first principle to merge together `{o,o'}` even if the patterns `s-v-o` and `s-v-o'` occur in two *different* relevant documents.

Another direction is to merge `{o,o'}` into one class if they co-occur with only *one* clausal argument in the same relevant document. For example, we could merge all objects co-occurring with a given verb in the document (irrespective of the subject), on the belief that the sense of the verb is fixed for the given discourse.

Thus, if we put in place some mechanisms for extending classes, ExDisco would have a better chance of recovering the missing documents. These variants seem potentially promising, and will require further research.

9.3 Research Problems

This section lists some of the important problems which remain to be resolved, but which lie beyond the scope of this thesis.

9.3.1 Variation across Scenarios

First, we clearly need to study the behavior of the discovery algorithms on additional scenarios to determine to what extent scenario-specific phenomena affect their performance. The two scenarios, "Management Succession" and "Mergers and Acquisitions", which we have evaluated on text filtering, exhibit a noticeable difference in the recall/precision plane.

On the other hand, on the "Natural Disasters" scenario we could not get reasonable text filtering scores at all. Despite the high-quality patterns which ExDisco finds on successive iterations, the bootstrapping does not seem to get off the ground to pull in more relevant documents. The best spread we could get with any seed was starting with recall 0.28 and precision 0.86, and after 270 iterations gets up to recall 0.41 (at 0.77 precision).

Clearly, a closely related question is how sensitive the discovery procedure is to the initial "seed" pattern set; i.e., how its success is dependent on the goodness of the patterns the user provides initially. However, in the case of "Natural Disasters", the difficulty may lie beyond the choice of a good seed. Our on-going work, [27] and [16] suggests that this may be due to two types of linguistic variation: differences in text type and the scale of discourse complexity inherent to a given scenario.

Similar work appeared earlier, e.g., [4], which began to address the important question of the necessity and possibility of qualifying the discourse-level complexity of a scenario. It would be very helpful if we could state in formal terms the basic intuition of anyone familiar with the MUC tasks, e.g., that the MUC-6 task was much "simpler" or tractable than the MUC-7 tasks.

In relation to the seed, another question is whether the seed can be something

other than a set of patterns, e.g., whether it may rather be a set of terms or concepts, or documents. Other research on unsupervised learning, e.g., [51], suggests that it's possible.

9.3.2 Convergence

We need to consider variants of the formulas for computing the confidence of a document with respect to patterns' ranks, and try to to determine empirically which of these variants is optimal.

We plan to experiment with selection of weights and thresholds for controlling the rankings of patterns and documents, criteria for terminating the iteration process, and for dynamic adjustments of these weights.

The fundamental question here is *convergence*, i.e., is there any sense in which we can hope or expect the discovery process to converge. At the time of its inception, the hope was that ExDisco would work for some number of iterations, and then come back to say that no more patterns can be discovered. Our experience so far shows that rather, as we saw section 8.2, (for those scenarios where we can get good results) ExDisco typically finds strong patterns, then reaches a critical point and lapses into accepting irrelevant patterns, after which the precision degrades.

The question is whether this is an artifact of the choice of scoring functions and weights, or whether divergence is somehow inherent in the task. It is important to investigate this and other unsupervised learning schemes, to try to find the answer. Perhaps a closer examination of the patterns around the critical point, might provide a clue. The hope would be to find a parameter which characterizes the likelihood of goodness on each iteration or indicates an appropriate place to

stop.

9.3.3 Syntactic Information

In the initial experiments with ExDisco we utilized only the main clausal arguments: subject, verb and object. However, the parser provides much richer information, (see section 7.2). We need to extend the procedures to take advantage of this information, generalizing on additional arguments.

The parser also provides information on noun phrases. In particular, we would expect that nominalizations are at least as important as verbs as indicators of relevance to a scenario. We need to extend the procedure to account for NP patterns as well.

There are also several problems of a more technical nature. One is providing more "help" to the parser. The parser routinely has trouble with certain common constructions which can be easily handled with simple semantic patterns; for example, the appositions "person, age", or compound locations "city, state". In a sense, these are not syntactic phenomena, and the procedure should not suffer because the parser misses them.

Often we find that a single triple has a strong score even without generalization (i.e., with no wild-card arguments). In such situations, the system "discovers" two or more of its generalizations, and gives the matched documents multiple credit. We need to find a disciplined way to compensate for this bias.

9.3.4 Person-in-the-Loop

An important path of investigation is "putting a person inside the loop", in keeping with a current trend in data mining research. We expect to be able to show *substantial* returns in some scenarios, for minimal intervention required at each iteration. The human may be asked to filter, say, the 5 top-ranking patterns, instead of just the single best one. S/he may also be asked to prune the suggested concept classes. We would expect improvements in precision on the successive iterations—this would be rather like having a growing human-selected, seed set of patterns and concepts, except that the user does not need to find the patterns, only to check the ones the system proposes.

We have begun investigating this direction, and added the basic functionality to ExDisco for human feedback and intervention at the end of each iteration. However, the initial results are inconclusive.

Several problems have emerged in connection with this extension. One problem is that the person in the loop in general does not wish to make a binary decision—the patterns are often neither good nor bad but somewhere in between. The system currently lacks a disciplined way of helping the user assign weight to the newly acquired pattern.

Another consideration is that at times ExDisco may find such strong, "killer" patterns that the user may wish to add them to the seed and restart the process from the beginning.

9.3.5 Putting It All Together

Lastly, we need to put in place mechanisms that would "glue" together phases II and III, PET and ExDisco. The latter discovers multiple useful example sentences to support the induction of a pattern into the pattern base. These examples could give rise to a wealth of modifiers for the given pattern.

As discussed in section 5.10, PET needs to be extended to incorporate the variation in modifiers into the pattern in a disciplined way. A useful "glue" tool would

- help the user organize the discovered examples,
- collapse the examples to find the largest common basic pattern, and then
- interact with the extended PET to enrich the pattern with an appropriate set of optional modifiers.

9.4 A Formal Framework

The presence of—by now—a considerable number of similar unsupervised approaches, as listed in section 2.3.4, suggests the presence of a fundamental commonality of principle. It would seem natural to seek to capture the commonality formally, in terms of features of the related approaches.

9.4.1 Co-Training

[5] makes an important contribution in this regard. The authors present a list of problems in the area of inducing classifiers from unlabeled data. They review the

corresponding solutions, which all involve un-supervised learning, and attempt to gather these approaches under a single umbrella, a unifying formal framework they call "co-training". According to Mitchell, the key defining characteristic of this class of problems is the presence of two or more "mutually redundant" views (or features) on the data-points which are being classified; the co-variance of the features with respect to the instances; and hence, the system's ability to progressively train multiple "independent" classifiers, starting with unlabeled data and a few labeled examples.

The authors develop the framework and present a PAC-based proof of learnability for problems falling into the class.

However, there are several unsatisfactory aspects to this view. Leaving aside for the moment the inherent violation of the "zero-one-infinity" principle, and leaving aside [11] (which indeed seems to match the framework), it does not seem natural to class most approaches we discussed in 2.3.4 under the all-encompassing heading of co-training.

9.4.2 Duality between Rules and Instances

it seems more intuitive to seek a simpler unifying principle in the learning approaches presented in section 2.3.4. Namely, the common shared feature of these approaches is the presence of a classifier on one hand, and an *instance* base on the other. The classifier may be multi-partite, but it is a single classifier in the sense that it learns to solve a single classification problem. In many cases, the classifier happens to be a set of rules whose number and confidence grows during training. The instance base is a sub-collection of data points which have acquired

classification labels during training. The presence of multiple features in the instances, and the co-variance of the features are still essential, since that is what allows bootstrapping to happen.

In this view, in the case of ExDisco, the *single* classifier is the growing set of patterns, and the instances are the documents. The classification label is the relevance judgement. In case of [1], [7] and [46], the classifier, again, is a growing set of patterns, and the instances are actual facts in the database, which find different textual realizations in different documents. In [29], training the "classifier" can be seen as expanding the set of "rules", which are translation word pairs. The instances are the aligned translation *sentence* pairs, in the given bi-lingual corpus. In [59], the rules are the collocations, and the instances are the discourses.

[5] cites in particular the algorithm of [59] as falling under the generalized "co-training" heading. One classifier is registered as in our analysis. The other classifier is the growing set of labeled documents—a suggestion which seems counter-intuitive. It is also difficult to accept the "mutually redundant, independently sufficient, multiple views" in this particular learning scheme.

It may be that some revised version of "co-training" would reflect the reality of these approaches more accurately. An important common feature may turn out to be the assignment of relevance rankings to the *instances* in the instance pool, rather than to rules only—an essential feature of ExDisco.

As a final comment, the heavy appeal to intuition in this section is itself an admitted indication of a need for more rigorous investigation.

9.5 Conclusion

The results presented in this thesis indicate that our method of corpus analysis can be used to rapidly identify a large number of relevant event-level patterns without a pre-classified training corpus.

We also feel that the techniques for generalization in pattern discovery, proposed in the present chapter, offer a potentially great opportunity for combating sparseness of data.

In conclusion we note that there is much room for further investigation. On the arts-and-crafts continuum, building effective IE systems remains—for now—somewhat closer to the former.

Appendix A

Sample Source Document

<DOC>

<DOCID> WSJ-demo.1 </DOCID>

<DOCNO> PetDemo00.0720 </DOCNO>

<HL> New Appointment:

@ Sample Patterns for PET

@ ----

@ By Roman Yangarber

@ Staff Reporter for the Proteus Project </HL>

<DD> 07/20/00 </DD>

<SO> WALL STREET JOURNAL (J), PAGE A16 </SO>

<TXT>

<p>

Information Resources Inc.'s London-based European Information Services operation has appointed George Garrick, 40 years old, as the new president.

<p>

He succeeds Mr. Sloane Talbot, who resigned, citing health reasons.

The decision came late yesterday.

<p>

</TXT>

</DOC>

Appendix B

Sample NE Output

<DOC>

<DOCID> WSJ-demo.1 </DOCID>

<DOCNO> PetDemo00.0720 </DOCNO>

<HL> New Appointment:

@ Sample Patterns for PET

@ ----

@ By <ENAMEX TYPE=PERSON>Roman Yangarber</ENAMEX>

@ Staff Reporter for the Proteus Project </HL>

<DD> 07/20/00 </DD>

<SO> WALL STREET JOURNAL (J), PAGE A16 </SO>

<TXT>

<p>

<ENAMEX TYPE=ORGANIZATION>Information Resources Inc.</ENAMEX>'s

<ENAMEX TYPE=LOCATION>London</ENAMEX>-based <ENAMEX

TYPE=ORGANIZATION>European Information Services</ENAMEX> operation has

appointed <ENAMEX TYPE=PERSON>George Garrick</ENAMEX>, 40 years old,
as the new president.
<p>
He succeeds Mr. <ENAMEX TYPE=PERSON>Sloane Talbot</ENAMEX>, who
resigned, citing health reasons. The decision came late yesterday.
<p>
</TXT>
</DOC>

Appendix C

Sample ST Output

```
<TEMPLATE-PetDemo00.0720-1> :=
    DOC_NR:              "PetDemo00.0720"
    CONTENT:             <SUCCESSION_EVENT-PetDemo00.0720-1>

<SUCCESSION_EVENT-PetDemo00.0720-1> :=
    SUCCESSION_ORG:      <ORGANIZATION-PetDemo00.0720-1>
    POST:                "president"
    IN_AND_OUT:          <IN_AND_OUT-PetDemo00.0720-1>
                         <IN_AND_OUT-PetDemo00.0720-2>
    VACANCY_REASON:      OTH_UNK
    OFFSETS:             ##273#419

<IN_AND_OUT-PetDemo00.0720-1> :=
    IO_PERSON:           <PERSON-PetDemo00.0720-1>
    NEW_STATUS:          IN
```

```
    ON_THE_JOB:          UNCLEAR

<IN_AND_OUT-PetDemo00.0720-2> :=
    IO_PERSON:           <PERSON-PetDemo00.0720-2>
    NEW_STATUS:          OUT
    ON_THE_JOB:          NO

<ORGANIZATION-PetDemo00.0720-1> :=
    ORG_NAME:            "European Information Services"   ##273#354#
    ORG_TYPE:            COMPANY
    ORG_LOCALE:          "London-based"   ##302#314#
    ORG_COUNTRY:         United Kingdom

<PERSON-PetDemo00.0720-1> :=
    PER_NAME:            "George Garrick"   ##369#383#

<PERSON-PetDemo00.0720-2> :=
    PER_NAME:            "Sloane Talbot"   ##440#453#
    PER_TITLE:           "Mr."
```

Bibliography

[1] Eugene Agichtein and Luis Gravano. Snowball: Extracting relations from large plain-text collections. In *Proceedings of the 5th ACM International Conference on Digital Libraries (DL'00)*, 2000. To appear.

[2] D. Appelt, J. Hobbs, J. Bear, D. Israel, M. Kameyama, and M. Tyson. SRI: Description of the JV-FASTUS System used for MUC-5. In *Proc. Fifth Message Understanding Conf. (MUC-5)*, Baltimore, MD, August 1993. Morgan Kaufmann.

[3] 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.

[4] Amit Bagga and Alan Biermann. Analyzing the performance of message understanding systems. Technical Report CS-1997-01, Dept. of Computer Science, Duke University, 1997.

[5] Avrim Blum and Tom Mitchell. Combining labeled and unlabeled data with co-training. In *Proceedings of the 11th Annual Conference on Computational*

Learning Theory (COLT-98), pages 92–100, New York, July 24–26 1998. ACM Press.

[6] Andrew Borthwick, John Sterling, Eugene Agichtein, and Ralph Grishman. Exploiting diverse knowledge sources via maximum entropy in named entity recognition. In *Proceedings of the Sixth Workshop on Very Large Corpora*, Montreal, Canada, August 1998.

[7] Sergey Brin. Extracting patterns and relations from the world wide web. In *WebDB Workshop at 6th International Conference on Extending Database Technology, EDBT'98*, 1998.

[8] M. E. Califf and R. J. Mooney. Relational learning of pattern-match rules for information extraction. In *Working Notes of AAAI Spring Symposium on Applying Machine Learning to Discourse Processing*, pages 6–11, Menlo Park, CA, 1998. AAAI Press.

[9] Mary Elaine Califf. *Relational Learning Techniques for Natural Language Information Extraction*. PhD thesis, Department of Computer Sciences, University of Texas, Austin, TX, August 1998. Also appears as Artificial Intelligence Laboratory Technical Report AI 98-276 (see http://www.cs.utexas.edu/users/ai-lab).

[10] Claire Cardie and David Pierce. Proposal for an interactive environment for information extraction. Technical Report TR98-1702, Cornell University, Computer Science, September 2, 1998.

[11] Michael Collins and Yoram Singer. Unsupervised models for named entity

classification. In *Proceedings of the Joint SIGDAT Conference on Empirical Methods in Natural Language Processing and Very Large Corpora*, College Park, MD, June 1999. University of Maryland.

[12] Ido Dagan, Shaul Marcus, and Shaul Markovitch. Contextual word similarity and estimation from sparse data. In *Proceedings of the 31st Annual Meeting of the Assn. for Computational Linguistics*, pages 31–37, Columbus, OH, June 1993.

[13] David Fisher, Stephen Soderland, Joseph McCarthy, Fangfang Feng, and Wendy Lehnert. Description of the UMass system as used for MUC-6. In *Proc. Sixth Message Understanding Conf. (MUC-6)*, Columbia, MD, November 1995. Morgan Kaufmann.

[14] Dayne Freitag and Andrew McCallum. Information extraction with HMMs and shrinkage. In *Proceedings of Workshop on Machine Learning and Information Extraction (AAAI-99)*, Orlando, FL, July 1999.

[15] The second Garnet compendium: Collected papers, 1990-1992. Technical Report CMU-CS-93-108, Carnegie Mellon University, Computer Science, February 1993.

[16] Michael Gregory. Private communication, 2000.

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

[18] Ralph Grishman. Tipster Phase II Architecture Design Document, Version 1.52. New York University, August 1995.

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

[20] Ralph Grishman, Catherine Macleod, and Adam Meyers. Comlex Syntax: Building a computational lexicon. In *Proc. 15th Int'l Conf. Computational Linguistics (COLING 94)*, pages 268–272, Kyoto, Japan, August 1994.

[21] Ralph Grishman, Catherine Macleod, and John Sterling. New york university: Description of the proteus system as used for muc-4. In *Proc. Fourth Message Understanding Conf. (MUC-4)*, pages 233–241, McLean, VA, June 1992.

[22] Ralph Grishman and John Sterling. New York University: Description of the PROTEUS System as used for MUC-5. In *Proc. Fifth Message Understanding Conf. (MUC-5)*, Baltimore, MD, August 1993. Morgan Kaufmann.

[23] Ralph Grishman and Roman Yangarber. Issues in corpus-trained information extraction. In *Proceedings of International Symposium: Toward the Realization of Spontaneous Speech Engineering*, pages 107–112, Tokyo, Japan, February 2000.

[24] Zellig S. Harris. Linguistic transformations for information retrieval. In *Proceedings of International Conference on Scientific Information*, 1957.

[25] Lynette Hirschman, Ralph Grishman, and Naomi Sager. Grammatically-based

automatic word class formation. *Information Processing and Management*, 11(1/2):39–57, 1975.

[26] Silja Huttunen. Private communication, 1999.

[27] Silja Huttunen. Private communication, 2000.

[28] Timo Järvinen and Pasi Tapanainen. A dependency parser for English. Technical Report TR-1, Department of General Linguistics, University of Helsinki, Finland, February 1997.

[29] Martin Kay and Martin Röscheisen. Text-translation alignment. *Computational Linguistics*, 19(1), 1993.

[30] W. Lehnert, C. Cardie, D. Fisher, J. McCarthy, E. Riloff, and S. Soderland. University of Massachusetts: MUC-4 test results and analysis. In *Proc. Fourth Message Understanding Conf.*, McLean, VA, June 1992. Morgan Kaufmann.

[31] Catherine Macleod, Ralph Grishman, and Adam Meyers. Creating a common syntactic dictionary of English. In *Proc. Int'l Workshop on Shared Natural Language Resources*, Nara, Japan, August 1994.

[32] Adam Meyers, Catherine Macleod, Roman Yangarber, Ralph Grishman, Leslie Barrett, and Ruth Reeves. Using NOMLEX to produce nominalization patterns for information extraction. In *Proceedings of the COLING-ACL '98 Workshop on Computational Treatment of Nominals*, Montreal, Canada, August 1998.

[33] George A. Miller. Wordnet: a lexical database for English. *Communications of the ACM*, 38(11):39–41, November 1995.

[34] Scott Miller, Michael Crystal, Heidi Fox, Lance Ramshaw, Richard Schwartz, Rebecca Stone, Ralph Weischedel, and the Annotation Group. Algorithms that learn to extract information; BBN: Description of the SIFT system as used for MUC-7. In *Proceedings of the Seventh Message Understanding Conference (MUC-7)*, Fairfax, VA, 1998.

[35] Tom Mitchell. The role of unlabeled data in supervised learning. In *Proceedings of the Sixth International Colloquium on Cognitive Science*, San Sebastian, Spain, 1999.

[36] *Proceedings of the Third Message Understanding Conference (MUC-3)*. Morgan Kaufmann, May 1991.

[37] *Proceedings of the Fourth Message Understanding Conference (MUC-4)*. Morgan Kaufmann, June 1992.

[38] *Proceedings of the Fifth Message Understanding Conference (MUC-5)*, Baltimore, MD, August 1993. Morgan Kaufmann.

[39] *Proceedings of the Sixth Message Understanding Conference (MUC-6)*, Columbia, MD, November 1995. Morgan Kaufmann.

[40] *Proceedings of the Seventh Message Understanding Conference (MUC-7)*, Fairfax, VA, 1998. `http://www.muc.saic.com/`.

[41] Johanna Nichols. Secondary predicates. *Proceedings of the 4th Annual Meeting of Berkeley Linguistics Society*, pages 114–127, 1978.

[42] Maria Teresa Pazienza, editor. *Information Extraction*. Springer-Verlag, Lecture Notes in Artificial Intelligence, Rome, 1997.

[43] Fernando Pereira, Naftali Tishby, and Lillian Lee. Distributional clustering of English words. In *Proceedings of the 31st Annual Meeting of the Assn. for Computational Linguistics*, pages 183–190, Columbus, OH, June 1993.

[44] Ellen Riloff. Automatically constructing a dictionary for information extraction tasks. In *Proceedings of Eleventh National Conference on Artificial Intelligence (AAAI-93)*, pages 811–816. The AAAI Press/MIT Press, 1993.

[45] Ellen Riloff. Automatically generating extraction patterns from untagged text. In *Proceedings of Thirteenth National Conference on Artificial Intelligence (AAAI-96)*, pages 1044–1049. The AAAI Press/MIT Press, 1996.

[46] Ellen Riloff and Rosie Jones. Learning dictionaries for information extraction by multi-level bootstrapping. In *Proceedings of Sixteenth National Conference on Artificial Intelligence (AAAI-99)*, Orlando, Florida, 1999.

[47] Naomi Sager, Carol Friedman, and Margaret Lyman. *Medical Language Processing: Computer Management of Narrative Data*. Addison Wesley, 1987.

[48] Yukata Sasaki. Applying type-oriented ILP to IE rule generation. In *Proceedings of Workshop on Machine Learning and Information Extraction (AAAI-99)*, Orlando, FL, July 1999.

[49] Stephen Soderland. Learning information extraction rules for semi-structured and free text. *Machine Learning*, 44(1-3):233–272, 1999.

[50] Tomek Strzalkowski and Jose Perez Carballo. Natural language information retrieval: TREC-4 report. In *Proc. Fourth Text Retrieval Conference*, Gaithersburg, MD, November 1995.

[51] Tomek Strzalkowski and Jin Wang. A self-learning universal concept spotter. In *Proceedings of 16th International Conference on Computational Linguistics (COLING-96)*, Copenhagen, August 1996.

[52] Pasi Tapanainen and Timo Järvinen. A non-projective dependency parser. In *Proceedings of the 5th Conference on Applied Natural Language Processing*, pages 64–71, Washington, D.C., April 1997. ACL.

[53] Cynthia A. Thompson, Mary Elaine Califf, and Raymond J. Mooney. Active learning for natural language parsing and information extraction. In *Proc. 16th International Conf. on Machine Learning*, pages 406–414. Morgan Kaufmann, San Francisco, CA, 1999.

[54] Roman Yangarber. PET: The Proteus Extraction Toolkit. User and Developer Manual, 1997.

[55] Roman Yangarber and Ralph Grishman. Customization of information extraction systems. In Paola Velardi, editor, *International Workshop on Lexically Driven Information Extraction*, pages 1–11, Frascati, Italy, July 1997. Università di Roma.

[56] Roman Yangarber and Ralph Grishman. NYU: Description of the Proteus/PET system as used for MUC-7 ST. In *MUC-7: Seventh Message Understanding Conference*, Columbia, MD, April 1998.

[57] Roman Yangarber, Ralph Grishman, Pasi Tapanainen, and Silja Huttunen. Automatic acquisition of domain knowledge for information extraction. In

Proceedings of the 18th International Conference on Computational Linguistics (COLING 2000), Saarbrücken, Germany, August 2000.

[58] Roman Yangarber, Ralph Grishman, Pasi Tapanainen, and Silja Huttunen. Unsupervised discovery of scenario-level patterns for information extraction. In *Proceedings of Conference on Applied Natural Language Processing (ANLP-NAACL'00)*, Seattle, WA, April 2000.

[59] David Yarowsky. Unsupervised word sense disambiguation rivaling supervised methods. In *Proceedings of the 33rd Annual Meeting of the Association for Computational Linguistics*, pages 189–196, Cambridge, MA, July 24–26 1995. ACM Press.