Proceedings of VEXTAL'99 - Venezia, San Servolo V.I.U. - 22-24 November 1999

A Named Entity Extraction System and its Web extensions

Fabio Rinaldi and Bill Black
Centre for Computational Linguistics, UMIST
PO Box 88, Manchester, M60 1QD, UK
$< fabio, bill >$ @$ccl.umist.ac.uk$

Abstract

In this work we describe a Named Entity Extraction system originally developed within the scope of the EU-funded FACILE project, and currently used within the CONCERTO project. The system has been firstly tested at the MUC-7 competition. The purpose of the system is to identify and classify proper names in free text. In the FACILE project these were mainly financial news, however the system and resources are not domain specific. Extensions to the resources are required only if new types of entities have to be captured. The architecture itself is language-neutral and resources have been developed for four European languages. Recent extensions include a Java-based client which allows access to the system via a conventional Web Browser.

1 Introduction

In many applications of Text Analysis, there is a clear need to identify reliably and tag proper names so that the further stages of analysis see them as a unit, and they can be given a semantic description. Not only it would be practically impossible to store all known proper names in a declarative resource like a lexicon, it would also fall short of the requirements of a working system, since this resource would become almost immediately out of date (as for example new company names are continuously created).

It is also useful to be able to identify when different tokens in the text corefer and often simple techniques of string matching (in particular for proper names) will be able to identify at least some of the coreferences (thus easing the task of a more sophisticated anaphora resolution module).

Capitalisation might help to identify the possible extent of proper names, but in English it applies to all classes of names and to nationality adjectives (which we may want to discount), as well as to sentence-initial words of any semantic class. Of course this may not always be available - we are treating text parts which are all capitalized, and we are also dealing with four languages, including German, where all nouns are capitalized.

Also potentially helpful is what is said about the named entities in the sentences in which they occur. For instances often in news, when an important topic entity is introduced, it comes with a *within-text description*. As an example consider the following sentence:

(1) *Prime* *Minister* Tony Blair

In this and similar sentences, the underlined word, forming the head of its descriptive noun phrases, refers to an occupational category that can only be filled by a human.

To identify instances of complex proper names, we need a language in which to specify their structure in terms of the tokens themselves and what we can find about them from resources such as a tagger, morphosyntactic analyser and database of known names and clue words. Each of these features needs to be coded so that they may be referred to in patterns/rules. In other words our metalanguage needs a syntax and a vocabulary. Beyond that, we have some additional requirements: we have to be able to attach scores to patterns so we can choose between them, and we also have to track coreferences among similar names.

In the example above, it is clearly possible to identify "Tony Blair" as a proper noun even if it doesn't belong to a list of known proper nouns, just because it occurs immediately after the clue word "Minister". Notice that a similar behaviour occurs with words such as "Doctor", "Professor", "Chief", etc. Therefore all these words could be given a single semantic tag (e.g. TITLE or OCCUPATION) which identifies them as clue words for a person. Once they have been tagged this way a rule like the following can be used to extract from the text instances of person names occurring after such clue words:

```
[syn=NP, sem = PER] =>  [sem = TITLE] \ [orth=C]+ / ;
```

This rule can be read as "if a sequence of capitalized words follows a word which has been tagged as a TITLE (clue word for a person), than that sequence of words is a NP identifying a person".

The work reported here has been originally developed in the context of the FACILE system, which categorises financial news and extracts information from them, in four European languages (see. [Ciravegna et al, 1999]). In FACILE the final stages of information extraction are done by analysing the full text using a typed-feature structured grammar, and with separate modules for anaphora resolution and template filling. The separate module for categorizing texts uses "shallow" pattern matching to identify the dominant themes.

These two modules use a common preprocessor, which tokenizes, tags and morphologically analyses texts, concluding by identifying and tagging complex proper names. This module is now been extended in the context of another EU-funded project (CONCERTO, see [Zarri et al, 1999] and [Bertino et al, 1999]). In this new context it is referred to as the Basic Semantic Element Extractor (BSEE) and this is the term that will be used in the rest of this article.

The components of the BSEE module may be pictured as in Figure 1. This presentation will focus on the description of the three main components of the BSEE module: the preprocessor (section 2), the DB lookup module (section 3) and the NE analyzer (section 4). In addition we will describe some recent work aimed at providing a user friendly Java-based interface (section 5).

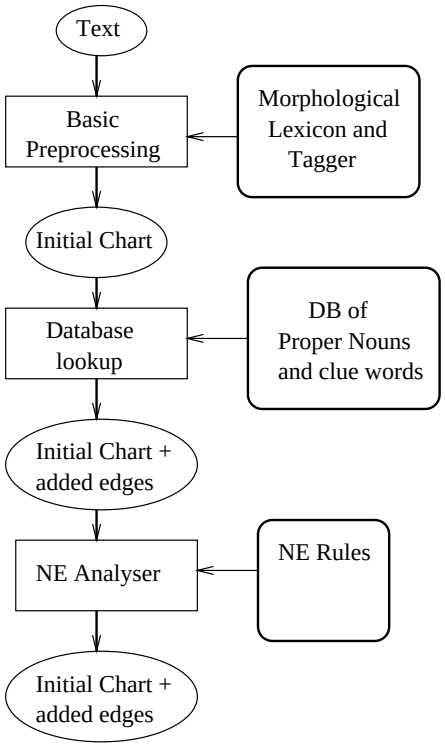

Figure 1: Architecture of the BSEE

2 Preprocessor

The preprocessor is responsible for the basic organization of the input text on which then the further modules can work. It provides the "vertical" organization (list of tokens), the "horizontal" organization (additional information about each single token), as well as the "structural" organization (zones).

Preprocessing involves tokenising the input string into units such as words, numbers, punctuation symbols, and filling out the chart columns for each token by consulting the morphological analyser and tagger for the language concerned.

Prior to tokenisation a text zoning component will identify conceptually distinct areas of the documents (e.g. title, abstract, body). The zone recognition is based on SGML/XML analysis. The input to the preprocessor must be a text encoded in Latin-1 and possibly containing SGML/XML markup. A declar-

ative specification allows the text zoner to deal with different DTDs in a flexible manner. Tokens, both single and multi-word tokens, are uniquely identified by a pair of string offsets. Each string offset pair consists of the start position of the token in the original text and the end position of the token.

The preprocessor invokes specialist components for lexical and morphological analysis. A tagger is used to select a single preferred reading. This may or may not be correct. Due to the fact that the Preprocessor employs both a tagger and a morphological component, the results of the tagger are used to select a 'good' morphological reading, in line with the syntactic tag, while the other morphological readings will still be available for recovery purposes.

Table 1: Attributes of a token

field	value	
start	28	char. offset
end	36	ditto.
zone	T	text zone
sep	040	octal char preceding
comp	C	compound?
orth	C	capitalised? etc.
token	"Reliance"	the string
norm	"Reliance"	root form
syn	NP	preferred syn tag
sem	()	cat. labels from database
good-morph	Nm+NP	morph. of SYN
other-morph	NIL	other poss. analyses
ante	NIL	antecedent if any

2.1 Multilingual Issues

The FACILE system was meant to work on four European languages: English, German, Italian and Spanish. Many multilingual applications, especially those developed within the framework of collaborative projects, use different technical approaches and processors for the different languages. In FACILE it was decided very early on to use generic components wherever possible in preference to language-specific alternatives. The morphological analysers all come from the same third-party source, and although they embody language-specific resources, they all use the same finite state technology, and all share an identical API.

This means in turn that the preprocessor is a single procedural component, which calls for a language-specific morphological analysis in a standard way, and then proceeds to do the same for the analysis of complex proper names and other named entities.

In English, the usage of a morphological analyzer might seem redundant (especially if the tagger provides a rich tag set), but the inflectional morphology of the other languages is more complex. For instance in German it is often the case that one word form receives three analyses differing only in the case feature - Akk, Dat or Nom. Different morphological analyses may involve also different normalized (i.e. root) forms (even if both readings are compatible with the PoS tag selected by the tagger).

Another well-known peculiarity of German at the morphological level is the way that noun compounds are constructed. Whereas English noun compounds are orthographically separated, the German ones are agglutinated. The possibility of having an ambiguity in the root form of a word, even if readings incompatible with the preferred PoS tag have been discarded, unfortunately percolates also to the level on which the subconstituents of a compound are represented.

The design of the preprocessor had therefore to be adapted to deal with these "worst cases" of ambiguity that the tagger cannot resolve.

2.2 The data structures

The results of the preprocessor phase are stored in a tabular structure, whose columns are indicated in Table 1. If several tokens from the text match a multi-word token in the database, the latter replaces the former.

Each row in the table describes the attributes of a single token. The column names and an example row are presented as the first two columns in Table 1. The table resembles the data structure used in chart parsing, except that there is only one initial edge per token. Alternative analyses are packed into the SEM and other-morph properties.

3 Database lookup

After the initial syntactic tagging and formatting it is necessary to perform a shallow semantic analysis of the text in order to allow the detection of compound

Proper Nouns and the extraction of named entities. As described in the introduction, often we can rely on some form of "clue word" in order to identify a named entity. So a first step in the analysis of a text is to identify those clue words. We have used for them simple semantic labels, which are abbreviations of the full semantic label used for the named entity (the labels are taken from the MUC competition tagset). A simple technique is to store those words in a DB together with their semantic label.

Database lookup may find information about semantic category or normalised form for words. This phase might also be responsible for the assignments of weights to those edges that contain a non-null value for SEM. Weights refer ONLY to the certainty of the semantic class being applicable, not to other ambiguities such as syntactic category. Where a term is found in the database, a new edge is added to the set of added edges. Where a word has more than one known semantic category, the alternatives are listed in the SEM field as a disjunction. In some cases the database may also return a NORM field containing a normalized value for the entity. For instance in some cases it might be helpful to have the numeric value corresponding to numbers (e.g. "thousand" => "1000") or the full form of an abbreviation ("MSc" => "Master of Science").

Database lookup incorporates an efficient mechanism for looking up sequences of tokens known as multi-word named entities. These are represented as added edges in the same way as simple tokens in the database. Where a multi-word token term is found in the database, it replaces the individual tokens it spans, and appears as one token.

It is possible to update the DB using a set of executable programs compiled from a C source or by means of a graphical tool, implemented using Tcl/Tk, which provides a user-friendly interface to the underlying C executables.

4 The Named Entity Analyzer

The NEA adds new chart edges for (sequences of) items it recognises as complete names or numbers or time expressions. It uses language-specific context-sensitive rules that can take account of any of the information placed in the chart by preceding modules to identify the extent and category of named entities etc. The NEA works like a chart parser in that it adds new edges without deleting any previous analyses. It also employs a weighting mechanism that enables it to either return all competing analyses or to select the most plausible one.

Standard pattern-matching languages like PERL, FLEX, SNOBOL etc., are designed to process patterns in text. Where tagged text is to be pattern-matched, it is straightforward to pair tags with words as in `the/AT cat/NN sat/VBD` etc. However, as we have pointed out, more than just the literal token and its syntactic tag are relevant to the NE recognition problem. To make all of these properties into facets of a token structure makes the statement of the rules in "raw" PERL hopelessly long-winded and error prone. One interesting effort at producing a higher level language that PERL is "Mother of PERL" (MOP) - see [Doran et al, 1997]. It enables the pattern writer to focus on a single or a few attributes at a time, but does this by the use of several separate layers of rules. Our language by contrast, allows conditions to refer to attributes arising from multiple levels of analysis in a single rule-set. However, in our view, a more significant advantage of our own rule language is its readability and accessibility to the rule-writer. We have in comparison far fewer symbolic operators and instead use a variant attribute-value matrix notation to make the individual rules more self-documenting.

Because in any one pattern constituent we would need to refer to only one or two of the properties, we chose to use a attribute-value notation for readability, but not one as powerful as we might want to use for a full syntactic and semantic analysis. Our language allows conditions to refer to attributes arising from multiple levels of analysis in a single rule-set. We have in comparison with other approaches far fewer symbolic operators and instead use a variant attribute-value matrix notation to make the individual rules more self-documenting.

In figure 2 two examples can be seen. Rule (A) is satisfied by a token whose normalised form (i.e. modulo capitalization) is "university," the literal "of" and a location or city name. A string matching this description is assigned the syntactic tag PN and the semantic tag ORG. Rule (B) illustrates the coreference operator >> (as well as including right and

left context), which stipulates that there be an antecedent constituent matching the AVM following the operator, which can satisfy the variable bindings established in the rest of the rule. In this case, both surname and title must be identical with their antecedents. The variable allows the repeatable part of an entity to be identified for matching with any subsequent mentions.

Unfortunately the limited space does not allow us to describe in detail these and other interesting characteristics of the rule language (e.g. scope control in rule application), for more details see Black et al (1998). The combination of all these features makes the rule language extremely powerful yet highly readable and accessible to the rule-writer.

(2)
```
(A) [syn=NP, sem=ORG] (0.9) =>
    \ [norm="university"],
      [token="of"],
      [sem=REGION|COUNTRY|CITY] / ;

(B) [syn=NP,sem=PER] (0.7) =>
     [sem=^PER,token=_T]
     \ [orth=C,token=_S] /
     [orth!=C]
     >> [sem=PER,surname =_S,title=_T];
```

4.1 Implementation

The current implementation of the rule interpreter is adapted from a left-corner chart parser, although we have considered compiling to finite state machinery. Some failsafe conditions prevent infinite loops which might be caused by uncarefully designed rules.

The working data structures of the interpreter include an active chart, comprising sets of active and inactive edges and a vertex index, and a property-value table which stores the values of any attributes not in the standard chart columns of Table 1.

As noted above, rules have a default certainty of 1, or an assigned certainty in the range -1 to 1. If rules give competing descriptions for the same span of the text, the `sem` value with the highest score is preferred. Where several rules come to the same conclusion about the `sem` value, evidence combination comes into play. We combine such scores using Shortliffe and Buchanan's Certainty Theory formula ([Shortliffe et al, 1975]). For reasons of space, we omit the details here.

Table 2: Scores

		Prec	Rec
training	english	94	92
	german	97	96
	italian	96	96
test	english	91	94
	german	94	85
	italian	86	92

After evaluation of certainties for each given text span, a further heuristic is applied which prefers longer spans to shorter in cases of overlap. The resulting analysis is a single semantic and property description of identified name expressions in the text.

At the end of the FACILE project a formal evaluation of its major components has been performed. The results obtained by the preprocessor are detailed in table 2 (also reported in [Ciravegna et al, 1999]). The evaluation was performed on two separate corpora of newswire messages (for each language) and the scores were obtained using the MUC scorer. Notice that the evaluation for Italian and German (as well as the development of the resources for those languages) was not performed by the authors of this paper.

5 Web Extensions

The main limitation of the system as developed within the scope of the FACILE project (as described in the previous sections) was the lack of an independent user interface. The Preprocessor and NE extraction modules, although central to the functioning of the FACILE integrated system, remained hidden behind our partners' components, which had independent user interfaces.

In order to allow us to exploit independently our system it was deemed necessary to develop a user interface. A natural approach was then to develop a Java client which would interact with the BSE system via a socket (as in [Black et al., 1999]).

Therefore recently we have produced a Web-based interface, implemented as a Java applet embedded within a web presentation of our current work. This allows a (possibly remote) user to interact with the system running as a server process on one of our de-

partmental machines. We have defined a special protocol that translates the user requests into commands to the server.

A potential problem with the implementation via socket discussed above is that it will not work if the remote user is behind a firewall (as that will prevent a socket connection from being established). As a workaround to this problem, we have also developed a different version of the client that connects via HTTP to a Java servlet running on our web server at the Department of Language Engineering. The servlet itself connects via a local socket to the BSE server and acts as a bridge mapping the socket protocol devised for the first version into HTTP.

6 Conclusions and further work

We have presented a flexible reusable component which performs preprocessing and Named Entity Extraction on free text. We have drawn attention to the main advantages of the notation in comparison with another language with a similar purpose, as well as with more general purpose languages.

We have also presented on-going work aimed at making the system available to non-technical users via a conventional Web Browser. This is due to be extended in the near future, while at the same time the declarative resources are undergoing a phase of revision and consolidation.

7 Acknowledgements

Former members of the team which designed and implemented the BSE module include Ralph Dressel and David Mowatt. Marta Saiz is currently revising the declarative resources and in particular the rule set for English. We are also grateful to all the members of the FACILE and CONCERTO projects for all their suggestions and contributions.

References

[Bertino et al, 1999] Bertino, E., Black, B., Brasher, A., Candela, V., Catania, B., Deavin, D., Esposito, F., McNaught, J., Persidis, A., Rinaldi, F., Semeraro, G., Zarri, G.P. Concerto, Conceptual Indexing, Querying and Retrieval of Digital Documents. *Proceedings of the ICMCS'99 Conference Special Event "THE EUROPEAN COMMUNITY DAY"*, June 7-11, 1999, Florence, Italy.

[Black et al, 1998] Black, W.J., Rinaldi, F. and Mowatt, D. (1998) FACILE: DESCRIPTION OF THE NE SYSTEM USED FOR MUC-7. In [MUC, 1998].

[Black et al., 1999] Black, W.J., Hill S. and Kassei M. (1999). In *Proceedings of the EACL workshop on Computer and Internet Supported Education in Language and Speech Technology*

[Chincor et al, 1993] Chinchor, N., Hirshman, L. and Lewis, D. (1993) Evaluating Message Understanding Systems. *Computational Linguistics* 19(3) 409-450.

[Ciravegna et al, 1999] Ciravegna, F., Lavelli, A., Mana, N., Matiasek, J., Gilardoni, L., Mazza, S., Ferraro, M., Black B., Rinaldi F. and Mowatt, D. FACILE: Classifying Texts with pattern matching and IE. *Accepted for publication at the IJCAI'99 Conference* August 3-6, 1999, Stockholm, Sweden

[Doran et al, 1997] Doran, C., Niv, M., Baldwin, B., Reynar, J.C. and Srinavas, B. (1997) Mother of PERL: A Multi-tier Pattern Description Language. *Proceedings of the International Workshop on Lexically Driven Information Extraction*, Frascati, Italy, July 16, 1997, 13–22.

[MUC, 1998] *Proceedings of 7th Message Understanding Conference* , Fairfax, VA. Available at http://www.muc.saic.com/

[Shortliffe et al, 1975] Shortliffe, E and Buchanan, B. (1975) "A model of inexact reasoning in medicine" *Mathematical Biosciences* **23**, 351-379

[Zarri et al, 1999] Zarri, G.P., Bertino, E., Black, B., Brasher, A., Candela, V., Catania, B., Deavin, D., Di Pace, L., Esposito, F., Leo, P., McNaught, J., Persidis, A., Rinaldi, F., Semeraro.,G. Concerto, An Environment for the 'Intelligent' Indexing, Querying and Retrieval of Digital Documents. *Proceedings of the ISMIS'99 Conference* Warsaw, Poland, June 8-11, 1999