XML tools and architecture for Named Entity recognition

Andrei Mikheev*, Claire Grover and Marc Moens
HCRC Language Technology Group,
University of Edinburgh,
2 Buccleuch Place, Edinburgh EH8 9LW, UK.
mikheev@harlequin.co.uk C.Grover@ed.ac.uk M.Moens@ed.ac.uk

November 26, 1998

Overview

Named Entity recognition involves identifying expressions which refer to (for example) people, organisations, locations, or artefacts in texts. This paper reports on the development of a Named Entity recognition system developed fully within the XML paradigm.

In section 1 we describe the nature of the Named Entity recognition task and the complexities involved. The system we developed was entered as part of a DARPA-sponsored competition, and we will briefly describe the nature of that competition.

We then give an overview of the design philosophy behind our Named Entity recognition system and describe the various XML tools that were used both in the development of the system and that make up the runtime system (section 2), and give a detailed description of how these tools were used to recognise temporal and numerical expressions (section 3) and names of people, organisations and locations (section 4). We conclude with a description of the results we achieved in the competition, and how these compare to other systems (section 5), and give details on the availability of the system (section 6).

1 Named Entity recognition

1.1 Named Entities

Named Entity recognition involves processing a text and identifying certain occurences of words or expressions as belonging to particular categories of Named Entities (NE). When this is done within the XML paradigm, the result is annotated text where each NE is annotated with information about the type of NE the system found.

Consider the following sentence:

*Now at Harlequin Ltd. (Edinburgh office)

```
On Jan 13th, John Briggs Jnr contacted Wonderful Stockbrokers Inc in
New York and instructed them to sell all his shares in Acme.
```

A Named Entity recognition system might annotate this sentence as follows:

```
On <NE TYPE="DATE">Jan 13th</NE>, <NE TYPE="PERSON">John Briggs Jnr</NE>
contacted <NE TYPE="COMPANY">Wonderful Stockbrokers Inc</NE> in
<NE TYPE="PLACE">New York</NE> and instructed them to sell all his shares in
<NE TYPE="COMPANY">Acme</NE>.
```

What counts as a Named Entity depends on the application that makes use of the annotations. One such application is document retrieval or automated document forwarding: documents annoted with NE information can be searched or forwarded more accurately than raw text. For example, NE annotation allows you to search for all texts that mention the company *Philip Morris*, ignoring documents about an unrelated person called Philip Morris. Or you can have all documents forwarded to you about a person called Gates, without receiving documents about things called gates. In a document collection annotated with Named Entity information you can easily find documents about the space shuttle Columbia without getting documents about Columbia D.C. Or you can retrieve all documents that talk about Hope (in Alabama), without also getting documents about people called Hope or about expectations and desires.

Another use of Named Entity recognition is in the construction of back-of-the-book indexes (e.g. an index for an encyclopaedia). In such an index you probably want to distinguish discussions of Alfred Nobel from mentions of people who won the Nobel prize, rather than just giving page numbers for every single occurence of "Nobel". This can be done if the NE recognition system has annotated mentions of Nobel as a person differently from mentions of Nobel as an artefact. Similarly, such an index will probably want to distinguish between Alzheimer the disease and Alzheimer the doctor, or between Java the programming language and Java the country.

Current work on metadata standardization (XML-Data, RDF) is concerned with the development of a syntax for annotating this kind of information. The system described here is intended to provide such annotation automatically.

1.2 Named Entities in the LTG system

We recently designed and built a Named Entity recognition system and entered the system in the Message Understanding Competition MUC. This is a competition on information extraction from text, sponsored by the U.S. Defense Advanced Research Projects Agency [8]. The Named Entities our system recognises and the type of annotation it uses for the markup are therefore the ones stipulated by the MUC competition rules. Here are some examples;

Temporal expressions. For the competition, absolute and relative temporal expressions needed to be marked up as `<TIMEX>` entities of type `DATE` or `TIME`. For example:

```
<TIMEX TYPE="DATE">all of 1987</TIMEX>
```

```
<TIMEX TYPE="DATE">from 1990 through 1992</TIMEX>
<TIMEX TYPE="DATE">first-half</TIMEX> profit
the <TIMEX TYPE="DATE">1986-87 academic year</TIMEX>
<TIMEX TYPE="TIME">8:24 a.m. Chicago time</TIMEX>
<TIMEX TYPE="TIME">early Friday evening</TIMEX>
<TIMEX TYPE="TIME">9 p.m.</TIMEX><TIMEX TYPE="DATE">Monday</TIMEX>
the <TIMEX TYPE="TIME">morning after the
                    <TIMEX TYPE="DATE">July 17</TIMEX> disaster</TIMEX>
on <TIMEX TYPE="DATE">All Saints' Day</TIMEX>
```

Mentions of currencies and percentages. Numeric expressions, monetary expressions and percentages, whether in numeric or alphabetic form, had to be marked up as `<NUMEX>` entities of type `MONEY` or of type `PERCENT`. For example:

```
<NUMEX TYPE="MONEY">175 to 180 million Canadian dollars</NUMEX>
<NUMEX TYPE="MONEY">10- and 20-dollar</NUMEX>bills
<NUMEX TYPE="MONEY">several million New Pesos</NUMEX>
the equivalent of less than <NUMEX TYPE="MONEY">a U.S. penny</NUMEX>
more than<NUMEX TYPE="PERCENT">95%</NUMEX>
```

Names of organisations, persons and locations. These are marked up as `<ENAMEX>` entities of type `ORGANIZATION`, `PERSON` or `LOCATION`. For example:

```
in <ENAMEX TYPE="LOCATION">North and South America</ENAMEX>
<ENAMEX TYPE="LOCATION">U.S.</ENAMEX> exporters
the <ENAMEX TYPE="ORGANIZATION">U.S. Fish and Wildlife Service</ENAMEX>
some <ENAMEX TYPE="ORGANIZATION">Treasury</ENAMEX> bonds and securities
the <ENAMEX TYPE="PERSON">Clinton</ENAMEX> government
<ENAMEX TYPE="ORGANISATION">Microsoft</ENAMEX> chairman
             <ENAMEX TYPE="PERSON">Bill Gates</ENAMEX> said yesterday...
```

Also, nicknames of organisations (e.g. "Big Blue"), locations (e.g. "the Big Apple") and people (e.g. "Mr. Fix-It") needed to be marked up as `ENAMEX` entities of the appropriate type.

1.3 The complexity of Named Entity recognition

Named Entity recognition is a difficult task for a number of reasons. First, the definition of what is and is not a Named Entity can be very complex. For example, according to the MUC competition rules, the following should *not* be marked up:

Artefacts. Artefacts like "the space shuttle Columbia" don't get marked up. The "Wall Street Journal" and "MTV" are organisations, and should be marked up as such. But when someone is *reading* the Wall Street Journal or *watching* MTV, they are artefacts, and should not be marked up. "Boeing" is an organisation, whose stocks may rise when Acme Corp orders another "Boeing". That second occurence of "Boeing" is an artefact and should not be marked up; but the first occurence of "Boeing" is an organisation and should be marked up.

Things named after people. "Nobel" and "Alzheimer" are names of people, and occurences of their names should be tagged as such. But in "Nobel Prize" or "Alzheimer's" their names should not be tagged.

Numbers which are not currencies or percentages. For example, one should not add markup to expressions like "unchanged at 95.05", "went up 12 points" or "1.5 times".

These rules may look *ad hoc*, but that is an accurate reflection of the nature of the Named Entity recognition task: what is and is not a Named Entity depends on the application that will make use of the Named Entities. The application may require you to distinguish Alfred Nobel from the Nobel prize, but need not. Also, in the system we developed we don't distinguish different types of artefacts—we only distinguish artefacts from organisations, people and locations, and leave the artefactual use of words like Boeing (the aircraft), Nobel (the prize) or Columbia (the space shuttle) unmarked. But one can easily imagine applications where transport vehicles (like a Boeing or a space shuttle) need to be marked separately from all other artefacts.

A second difficulty is that it is important to tag exactly the right words. The entire string "Arthur Andersen Consulting" should be marked as an `ORGANIZATION`; one should not mark the substring "Arthur Andersen" as a `PERSON`. In "Canada's Parliament", "Canada" (without the 's) should be marked up as `LOCATION`; "Parliament" should be marked up as `ORGANIZATION`. Again, this may appear *ad hoc* and the definition of how much should be marked up will be defined by the application. But for any application, consistency of NE markup, however *ad hoc* it may seem, is crucial.

The third and biggest problem is that Named Entities are expressed with words which can refer to many other things. One might think that Named Entity recognition could be done by using lists of (e.g.) names of people, places and organisations, but that is not the case. To begin with, the lists would be huge: it is estimated that there are 1.5 million unique surnames just in the U.S. [11]. It is not feasible to list all possible surnames in the world in a Named Entity recognition system.

There is a similar problem with company names. A list of all current companies worldwide would be huge, if at all available, and would be out of date tomorrow since new companies are formed all the time. In addition, company names can occur in variations: a list of company names might contain "The Royal Bank of Scotland plc", but that company might also be referred to as "The Royal Bank of Scotland", "The Royal" or "The Royal plc". These variations would all have to be listed as well.

But even if it was possible to list all possible organisations and locations and people, there would still be the problem of overlaps between the list. Names such as Emerson or Washington could be names of people as well as places; Philip Morris could be a person or an organisation. In addition, such lists would also contain words like "Hope" (a location) and "Thinking Machines" (a company), whereas these words could also occur in contexts where they don't refer to named entities. One could add some intelligence to the system and only tag these words when they have a capital letter. But that would still lead to erroneous markup when "Hope" occurs at the start of a sentence, or when "Thinking Machines" occurs in an all-capitalised headline. ‘

Identifying temporal expressions seems easier—after all, there are only 12 months, and we can list these and reliably identify them. But a system that does this might get confused when it finds a mention of "the Chinese-built Long March rocket" or a reference to someone called "April May", expressions which obviously should not be marked up as dates.

1.4 The MUC Competition

The MUC competition for which we built our system took place in March 1998. Prior to the competition, participants received a detailed coding manual which specified what should and should not be marked up, and how the markup should proceed. They also received a few hundred articles from the New York Times Service, marked up by the organisers according to the rules of the coding manual.

For the competition itself, participants received 100 articles. They then had 5 days to perform the chosen information extraction tasks (in our case: Named Entity recognition) without human intervention, and markup the text with the Named Entities found. The resulting marked up file then had to be returned to the organisers for scoring.

Scoring of the results is done automatically by the organisers. The scoring software compares a participant's answer file against a carefully prepared key file; the key file is considered to be the "correctly" annotated file. Amongst many other things, the scoring software calculates a system's recall and precision scores:

Recall: Number of correct tags in the answer file over total number of tags in the key file.

Precision: Number of correct tags in the answer file over total number of tags in the answer file.

Recall and precision are generally accepted ways of measuring system performance in this field. For example, suppose you have a text which is 1000 words long, and 20 of these words express a location. Now imagine a dumb system which assigns the `LOCATION` tag to every single word in the text. This system will have tagged correctly all 20 locations, since it tagged everything as `LOCATION`; its recall score is 20/20, or 100%. But of the 1000 `LOCATION` tags it assigned, only those 20 were correct; its precision is therefore only 20/1000, or 2%.

Here is an invented example of the kind of text the participants in the MUC competition had to process. The reason for inventing an example is that it allows us to demonstrate a wider range of phenomena in a more compact way:

```
<DOC>
<PREAMBLE>
GENERAL TRENDS ANALYST PREDICTS LITTLE SPRING EXPLOSION
By Liza McDonald
</PREAMBLE>
<TEXT>
<P>Flavel Donne Jr, an analyst with General Trends Inc, announced 2 days ago that
Little Spring would come to a loud end on May 29, 1999. General Trends, which is
based in Little Spring, has been producing predictions like this since early 1963.</P>
<P>Donne is C.E.O. of General Trends and also of Adam Kluver Ltd. But John May, 29,
```

```
spokesman for Adam Kluver, said yesterday they distanced themselves from Donne's
prediction. He added that their stock had gone down 12% since May and is now valued
at 130 million Canadian dollars. Flavel Donne was 42 last Thursday.</P>
</TEXT></DOC>
```

The example was constructed to illustrate a wide range of phenomena:

- Company names are fictitious, and not part of any lists of existing company names.
- Company names are multi-word expressions, which contain common words (general, trends) or which look like person names (Adam Kluver).
- Company names are sometimes referred to only in part: "Adam Kluver Inc." is also referred to as Adam Kluver, which could be mistaken for a person; "General Trends Ltd" is also referred to as "General Trends", which—especially in the capitalized headline—could be mistaken as a common noun phrase (an analyst of general trends).
- Person names have unusual christian names (Flavel, which we invented and is unlikely to be in any list of Christian names) or possibly confusing surnames (May, which could be mistaken for a temporal expression).
- There are multi-word person names ("Flavel Donne Jr"), but the same person is also referred to as just "Donne".
- The text contains dates, percentages and monetary values, which should be tagged. It also contains other numbers, which should not be tagged: in "Donne is 42", the number should not be tagged; in "2 days ago", the "2" should not be tagged, but the whole expression should be tagged as a temporal expression.
- In one instance, "May" followed by a number indicates a date, in another it indicates the name of a person followed by an age. This should result in different markup.

Our MUC system produces the following output:

```
<DOC>
<PREAMBLE>
<ENAMEX TYPE='ORGANIZATION'>GENERAL TRENDS</ENAMEX> ANALYST PREDICTS
<ENAMEX TYPE='LOCATION'>LITTLE SPRING</ENAMEX> EXPLOSION
By <ENAMEX TYPE='PERSON'>Liza McDonald</ENAMEX>
</PREAMBLE>
<TEXT>
<P>
<ENAMEX TYPE='PERSON'>Flavel Donne Jr</ENAMEX>, an analyst with
<ENAMEX TYPE='ORGANIZATION'> General Trends Inc</ENAMEX>, announced
<TIMEX TYPE='DATE'>2 days ago</TIMEX> that
<ENAMEX TYPE='LOCATION'>Little Spring</ENAMEX> would come to a loud end
on <TIMEX TYPE='DATE'>May 29, 1999</TIMEX>.
<ENAMEX TYPE='ORGANIZATION'>General Trends</ENAMEX>, which is based in
<ENAMEX TYPE='LOCATION'>Little Spring</ENAMEX>, has been producing predictions like
this since <TIMEX TYPE='DATE'>early 1963</TIMEX>.
<P>
```

```
<ENAMEX TYPE='PERSON'>Donne</ENAMEX> is C.E.O. of <ENAMEX TYPE='ORGANIZATION'>General
Trends</ENAMEX> and also of <ENAMEX TYPE='ORGANIZATION'>Adam Kluver Ltd.</ENAMEX>
But <ENAMEX TYPE='PERSON'>John May</ENAMEX>, 29, spokesman for
<ENAMEX TYPE='ORGANIZATION'>Adam Kluver</ENAMEX>, said
<TIMEX TYPE='DATE'>yesterday</TIMEX> they distanced themselves from
<ENAMEX TYPE='PERSON'>Donne</ENAMEX>'s prediction. He added that their stock had
gone down <NUMEX TYPE='PERCENT'>12%</NUMEX> since <TIMEX TYPE='DATE'>May</TIMEX>
and is now valued at <NUMEX TYPE='MONEY'>130 million Canadian dollars</NUMEX>.
<ENAMEX TYPE='PERSON'>Flavel Donne</ENAMEX> was 42 <TIMEX TYPE='DATE'>last Thursday</TIMEX>.
</TEXT>
</DOC>
```

2 LTG text handling tools

2.1 SGML awareness

At the Language Technology Group we have developed a suite of reusable text processing tools. These are modular tools with stream input/output; each tool does a very specific job, but can be combined with other tools in a pipeline. Different combinations of the same tools can thus be used in a pipeline for completing different text processing tasks.

Our architecture imposes an additional constraint on the input/output streams: they should have a common *syntactic* format. For this common format we use eXtensible Markup Language (XML).

A tool in our architecture is thus a piece of software which uses an API for all its access to XML data and performs a particular task: exploiting markup which has previously been added by other tools, removing markup, or adding new markup to the stream(s) with or without removing the previously added markup. This approach allows us to remain entirely within the XML paradigm during text processing. At the same time, we can be very general in the design of our tools, each of which can be used for many different purposes. Furthermore, because we can pipe data through processes, the UNIX operating system itself provides the natural "glue" for integrating data-level applications.

The XML-handling API in our workbench are our LT NSL and LT XML libraries ([12], [13]). They allow a tool to read, change or add attribute values and character data to XML elements and to address a particular element in an XML stream using a query language called *ltquery*.

ltquery offers a way of specifying particular nodes in the XML document structure. For example, the newspaper articles we were dealing with in the MUC competition can be represented as the SGML tree illustrated in Figure 1.

Queries in *ltquery* are coded as strings which give a (partial) description of a path from the root of the XML document (the top-level element) to the desired XML element(s). For example, the query

```
.*/TEXT/.*/S[STATUS="PARSED"]
```

refers to any `<S>` element whose attribute `STATUS` has the value `PARSED` and which occurs at any level of nesting inside a `<TEXT>` element which, in turn, can occur anywhere inside the

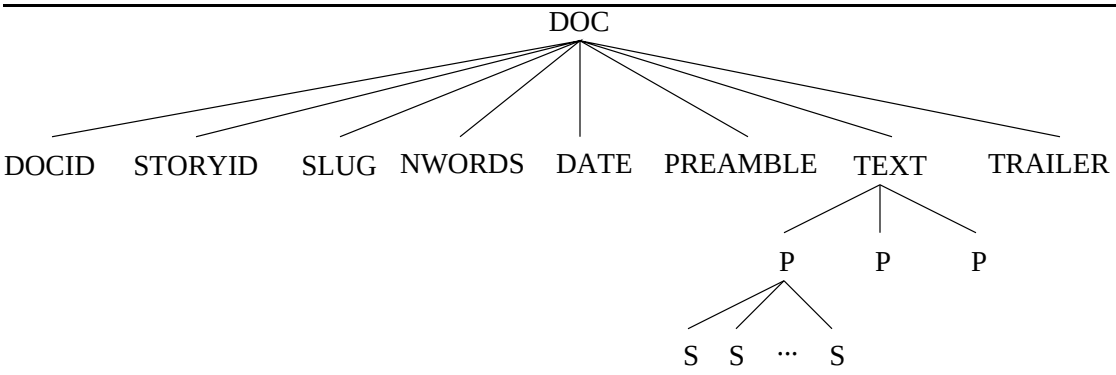

Figure 1: Partial SGML tree for a MUC Newspaper article

document's top-level element. It does not apply, e.g., to `<S>` elements inside the documents `<PREAMBLE>`.

The example shows that an *ltquery* query is a sequence of terms, separated by slashes. Each term in the query describes either an XML element or a nested sequence of XML elements. Element names can be followed by a list of attribute specifications in square brackets. An item that ends in a `*` matches a nested sequence of zero or more XML elements, each of which match the item without the `*`. For example, `P*` will match a `<P>` element, arbitrarily deeply nested inside other `<P>` elements. A full stop will match any XML element name; thus, a simple way of finding a `<P>` element anywhere inside a document is to use the query `.*/P`.

A condition with an index n matches only the nth sub-element of the enclosing element. Index counting starts from 0. Thus, `DOC/TEXT/P[0]` will give all first paragraphs under `<TEXT>` elements which are under `<DOC>`.

The simplest way of configuring our XML tools is to specify in a query where the tool should apply its processing. Using the syntax of *ltquery* we can directly specify which parts of the stream we want to process and which parts we want to skip. This also allows us to provide a tool with processing resources (e.g. grammars) specifically tailored to those document parts the tool is attending to. For example, we have a tool called `fsgmatch` which can be used to identify certain SGML elements in the input text and wrap them into larger SGML elements, according to rules specified in resource grammars. It can be called with different resource grammars for different document parts. Here is an example pipeline using `fsgmatch`:

```
>> cat text | fsgmatch -q ".*/DATE|NWORDS" date.gr
            | fsgmatch -q ".*/PREAMBLE"  preamb.gr
            | fsgmatch -q ".*/TEXT/P[0]"  first.gr
```

In this pipeline, `fsgmatch` takes the input text, and processes the data that has been marked up as `<DATE>` or `<NWORDS>` using a resource grammar called `date.gr`; then it processes the data in `<PREAMBLE>` using the resource grammar `preamb.gr`; and then it processes the first paragraph in the `<TEXT>` section using the grammar `first.gr`.

This technique allows one to tailor resource grammars very precisely to particular parts of the text. For example, the reason for applying `first.gr` to the first paragraph of a newspaper

article is that that paragraph often contains unusual information which occurs nowhere else in the article in that form. Here is the start of a typical article:

```
CAPE CANAVERAL, Fla. &MD;  Working in chilly temperatures Wednesday...
```

In our analysis of the MUC newspaper articles, we noticed that if an article starts with capitalized words followed by `&MD;` the capitalized words indicate a location. It is easy to capture this in a grammar. But the phenomenon only occurs in text initial `<P>` elements. And it is very efficient to be able to tell `fsgmatch` only to apply that specialised grammar to the first `<P>` element of any text it is processing.

We have developed a range of SGML and XML-aware processing tools. Some of them are low-level tools, such as `sgdelmarkup` which strips unwanted markup from a document, or `sgsed` and `sgtr`, which are SGML-aware versions of the UNIX tools `sed` and `tr`; some are higher-level tools, such as the SGML transducer `fsgmatch` mentioned above. Combinations of these tools provide us with the means to explore large text collections and to do fast prototyping of text processing applications. We have used these tools in the development of systems for many different applications, such as statistical text categorization [2], information extraction in a medical domain [3], collocation extraction for lexicography [1], etc. A detailed description of the tools, their interactions and applications can be found in [4] and [10]; information can also be found at our website, `http://www.ltg.ed.ac.uk/software/`. In the rest of this section, we will concentrate on some of the higer-level SGML-aware tools used in the Named Entity recognition system.

2.2 lttok

`lttok` is an SGML-aware tokeniser. Tokenisers take an input stream and divide it up into words or tokens, according to some agreed definition of what a token is. This is not just a matter of finding white spaces between characters. For example, one needs to decide whether "I've" and "can't" are one or two tokens. Also, for some applications one may want to treat as one token multi-word expressions like "Tony Blair Jnr", "President Bill Clinton", "Mr de Toqueville" or "January 17th, 1998". And hyphenated words like "first-quarter-charge" can be treated as a single token or three tokens, depending on the application.

The LTG tokeniser `lttok` works at the character level: it looks at the characters in the input stream and, using finite-state machinery, bundles them into tokens according to rules specified in its resource grammars. The input to `lttok` can be SGML-marked up text, and `lttok` can be directed to only process parsed character data within certain SGML or XML elements.

Here is an example of the use of `lttok`:

```
cat text | lttok -q ".*/P|TITLE|PREAMBLE|TRAILER" -mark W -attr C standard.gr
```

`lttok` tokenises the character data in all the `<P>` elements as well as in the `TITLE`, the `PREAMBLE` and the `TRAILER`, using the rules in the resource grammar `standard.gr`. The tokens it finds will be marked up using the SGML element `<W>`, and attribute information will be added using the attribute name `C`. The resource file stipulates what the possible values are for this attribute. Here is some example output from this pipeline:

```
s <W C='W'>Donne</W> <W C='W'>is</W> <W C='W'>C.E.O.</W> <W C='W'>of</W> <W
C='W'>General</W> <W C='W'>Trends</W> <W C='W'>and</W> <W C='W'>also</W>
<W C='W'>of</W> <W C='W'>Adam</W> <W C='W'>Kluver</W> <W C='W'>Ltd.</W> <W
C='W'>But</W> <W C='W'>John</W> <W C='W'>May</W><W C='CM'>,</W> <W C='CD'>29</W><W
C='CM'>,</W> <W C='W'>spokesman</W>...
...  <W C='W'>is</W> <W C='W'>now</W> <W C='W'>valued</W> <W C='W'>at</W> <W
C='CD'>130</W> <W C='W'>million</W> <W C='W'>Canadian</W> <W C='W'>dollars.</W>
...
```

Because of instructions in the resource file `standard.gr`, `lttok` also added the attribute `C` to each `<W>` element, whose value is `W` in the case of a word, `CM` in the case of a comma, `CD` in the case of a numeral, etc. This is information which other processing tools can make use of.

2.3 ltstop

As the above example shows, although the tokeniser adds annotation for commas, it does not add annotation for full stops. The reason for this is that not every period is a full stop; some are part of an abbreviation. Depending on the choice of resource file for `lttok`, a period will either always be attached to the preceding word (as in the above example, where the full stop stays with the sentence-final word "dollars" and with the abbreviation "C.E.O.") or it will always be split off.

This creates an ambiguity where a sentence-final period is also part of an abbreviation, as in our example "...and also of General Trends Ltd. But..." For many reasons is it useful to know where a sentence ends, and looking for a full stop followed by a space and a capital letter is not always sufficient, as illustrated in "It is the B.B.C. Secretary-General who..."

To resolve this ambiguity we use a special program, `ltstop`, which applies a maximum entropy model pre-trained on a corpus [7]. The statistical model knows which features are relevant in deciding whether a word is an abbreviation (e.g. usual length of abbreviations, capitalization, preceding words, ...) or when a word is sentence-final, or both. It has acquired these features automatically, on the basis of a corpus in which abbreviations and full-stops have been hand-annotated.

In the above example, `ltstop` will split the period from ordinary sentence-final words and create an end-of-sentence token `<W C=".">.</W>`; or it will leave the period with the word if it is an abbreviation; or, in the case of sentence-final abbreviations, it will leave the period with the abbreviation and in addition create a virtual full stop `<W C="."></W>`

Like the other LTG tools `ltstop` can be targeted at particular SGML elements. In our example, we want to target it at `<W>` elements within `<P>` elements—the output of `lttok`. It can be used with different maximum entropy models, trained on different types of corpora.

For our example, the full pipeline looks as follows:

```
cat text | lttok -q ".*/P|TITLE|PREAMBLE|TRAILER" -mark W -attr C standard.gr
         | ltstop -q ".*/P/W"  fs_model.me > text.stop
```

This will generate the following output in `text.stop`:

```
<W C='W'>Donne</W> <W C='W'>is</W> <W C='W'>C.E.O.</W> <W C='W'>of</W> <W
C='W'>General</W> <W C='W'>Trends</W> <W C='W'>and</W> <W C='W'>also</W> <W
C='W'>of</W> <W C='W'>Adam</W> <W C='W'>Kluver</W> <W C='W'>Ltd.</W><W C='.'></W>
<W C='W'>But</W> <W C='W'>John</W> ...
...  <W C='W'>is</W> <W C='W'>now</W> <W C='W'>valued</W> <W C='W'>at</W> <W
C='CD'>130</W> <W C='W'>million</W> <W C='W'>Canadian</W> <W C='W'>dollars</W><W
C='.'>.</W> ...
```

Note how `ltstop` left periods with abbreviations like "C.E.O.", separated off the full stop after "dollars", and left the period with "Ltd." but added a final stop to this sentence, making explicit that the period after "Ltd." has two distinct functions.

2.4 ltpos

Another standard LTG tool we use in our MUC system is our part-of-speech tagger `ltpos` [6]. Part-of-speech tagging (POS tagging) involves annotating words (as identified by the tokeniser) with information as to whether they are a verb, a noun, etc. To do this, POS taggers look up words in a lexicon which will tell them that, e.g., "left" is most likely to be a past tense verb (as in "he left") or an adjective ("my left foot"), but could also be a past participle ("they have left"), a noun ("on the left"), or an adverb ("go left"). Taggers also have statistical co-occurrence information, e.g. that an adjective is more likely to be followed by a noun than by a verb.

Part-of-speech tagged text is useful input for a Named Entity recognition system. For example, in "GIVE ME THE BILL", "BILL" will be tagged as a noun; in "GIVE ME BILL", "BILL" will be tagged as a proper name. The theoretical difference between a noun and a proper name is not important for present purposes, except that names of people tend to be proper names rather than nouns. On the basis of this information, a Named Entity recognition system can decide that "BILL" in the first sentence is more unlikely to be a `<PERSON>` than in the second sentence. This is obviously not sufficient to make a decision either way as to what sort of named entity "BILL" is, but it provides some extra evidence which can be used in combination with (for example) contextual clues.

Our part-of-speech tagger `ltpos` is SGML-aware: it reads a stream of SGML elements specified by the query and applies a statistical technique to assign the most likely POS tags. An important feature of the tagger is an advanced module for handling words which are not in the lexicon [5]. This proved to be crucial for name spotting: given that part-of-speech information can be a great help in detecting names, the POS tagger needs to be able to POS~tag unknown words—like the word "Donne" in "Donne is 42".

`ltpos` also carries out a few other tasks which are useful for Named Entity recognition. For capitalised words, `ltpos` adds information as to whether the word exist in lowercase in the lexicon (marked as `L="l"`) or whether it exists in lowercase elsewhere in the same document (marked as `L="d"`), or none of the above (marked as `L="#"`). This information is particularly useful for multi-word Named Entities, which contain common words: suppose a text contains the sentence "Suspended Ceiling Contractors Ltd denied the charge". Since the sentence-initial word has a capital letter, it could be an adjective modifying the company "Ceiling Contractors Ltd", or it could be part of the company name, "Suspended Ceiling Contractors Ltd". By marking early on that "suspended" also occurs in the lexicon in lowercase, the system will later know to be cautious about how many words to include in the `<ORGANIZATION>`

tag.

This is what the pipeline looks like:

```
cat text.tok | ltpos -q ".*/P|PREAMBLE|TRAILER|BC|TITLE"
                              -pos_attr C -lookup_attr L posgram > text.pos
```

The call to `ltpos` specifies that the part of speech tags should be entered as values to the attribute `C`; in other words, it changes the current `W` values of the `C` attribute to POS values. POS values are reasonably mnemonic abbreviations, fairly standard in the computational linguistics literature—such as JJ for adjective, CC for conjunction, NN for singular noun, NNP for singular proper name, and DT for determiner. The pipeline gives the following output:

```
<W L='#' C='NNP'>Flavel</W> <W L='#' C='NNP'>Donne</W> <W L='#' C='NNP'>Jr</W><W
C=','>,</W> <W C='DT'>an</W> <W C='NN'>analyst</W> <W C='IN'>with</W> ...
```

2.5 fsgmatch

The core tool in our MUC system is **fsgmatch**. **fsgmatch** is an SGML transducer. It takes certain types of SGML elements and wraps them into larger SGML elements. In addition, it is also possible to use **fsgmatch** for character-level tokenisation, but in this paper we will only describe its functionality at the SGML level.

fsgmatch can be called with different resource grammars, e.g. one can develop a grammar for recognising names of organisations or temporal expressions. Like the other LTG tools, it is possible to use **fsgmatch** in a very targeted way, telling it only to process SGML elements within certain other SGML elements, and to use a specific resource grammar for that purpose.

The combined functionality of **lttok** and **fsgmatch** gives system designers many degrees of freedom. Suppose you want to map character strings like "25th" or "3rd" into SGML entities. You can do this at the character level, using **lttok**, specifying that strings that match `[0-9]+[-]?((st)|(nd)|(rd)|(th))` should be wrapped into the SGML structure `<W C="ORD">`. Or you can do it at the SGML level: if your tokeniser had marked up numbers like "25" as `<W C="CD">` then you can write a rule for **fsgmatch** saying that `<W C="NUM">` followed by a `<W>` element whose character data consist of **th**, **nd**, **rd** or **st** can be wrapped into an `<W C="ORD">` element.

A transduction rule in **fsgmatch** can access and utilize any information stated in the element attributes, check sub-elements of an element, do lexicon lookup for character data of an element, etc. For instance, a transduction rule can say: "if there are one or more `W` elements (i.e. words) with attribute `C` (i.e. part of speech tag) set to NNP (proper noun) followed by a `W` element with character data "Ltd.", then wrap this sequence into an `ENAMEX` element with attribute `TYPE` set to `ORGANIZATION`.

Transduction rules can check left and right contexts, and they can access sub-elements of complex elements; for example, a rule can check whether the last `W` element under an `NG` element (i.e. the head noun of a noun group) is of a particular type, and then include the whole noun group into a higher level construction. Element contents can be looked up in a lexicon. The lexicon lookup supports multi-word entries and multiple rule matches are always resolved to the longest one.

An example of a small but useful thing we use `fsgmatch` for is to assign certain "semantic" tags which are particularly useful for MUC processing. For example, words ending in *-yst* and *-ist* (analyst, geologist) as well as words occurring in a special list of words (spokesman, director) are recognised as professions and marked as such (`S="PROF"`). Adjectives ending in *-an* or *-ese* whose root form occurs in a list of locations (American, Japanese) are marked as locative adjectives (`S="LOC_JJ"`).

To achieve this, it makes most sense to invoke `fsgmatch` immediately after `ltpos`:

```
cat text.pos | fsgmatch  -q ".*/P|PREAMBLE|TRAILER" sem.gr

<W L='#' C='NNP'>Flavel</W> <W L='T' C='NNP'>Donne</W> <W L='#'
C='NNP'>Jr</W></ENAMEX><W C=',',>,</W> <W C='DT'>an</W> <W S='PROF'
C='NN'>analyst</W> <W C='IN'>with</W>
```

Because `fsgmatch` plays such a crucial role in our MUC system, we describe it and the rules in the resource grammars in more detail in the following section.

3 TIMEX, NUMEX

Temporal and numerical expressions in English newspapers have a fairly structured appearance which can be captured by means of grammar rules. We developed a grammar for the temporal expressions we needed to capture. We also compiled lists of temporal entities, like days of the week and names of months (including abbreviations), and holidays and festivals (like "Hannukah" and "Hogmanay"). We also compiled a grammar of numerical expressions, as well as a list of currencies. The SGML transducer `fsgmatch` uses these resources to wrap the appropriate strings with `timex` and `numex` tags.

Figure 2 is an excerpt of the kind of resource file used by `fsgmatch` to identify certain `timex` expressions in the texts.

One of the rules in Figure 2 is called `day-name`. Its type is `DISJF`, which means that, for the rule to be successful, one of its subrules (`day-name-full` or `day-name-abbrev`) should succeed.

The rule `day-name-full` checks whether the input matches `CCAPWRD`—it checks if the input is an SGML entity labelled `<W>` (i.e. a word), whose PCDATA match the regular expression given in the entity definition for `CCAPWRD` (i.e. whether it is a capitalized word). When it finds a matching SGML item, it checks whether this word also occurs in the file `TIM_lex`—a file containing many temporal expressions, such as Monday, January, Tue, and Hogmanay, with tags indicating whether they are days of the week, holidays, etc. If the capitalized word is found in that file, its tag is checked. If the tag is found to be `DY` the `<W>` element is wrapped in a `<TIMEX>` element of type `DATE`.

It is worth pointing out that the resource files of which Figure 2 is a small excerpt are themselves structured as XML documents. We firmly believe that a good strategy for building text processing applications like the NE system is to build them using XML annotated stream input/output, but it does not follow from this that all the resource files that are called in the course of this process should also be in XML. However, because the production of the resource

```
<?XML version="1.0"    ?>
<!DOCTYPE RULES SYSTEM "RuleSpec.dtd" [
<!ENTITY CAPWRD      "W/#~^[A-Z][A-z]*$" >
]>

<RULES  name="date" apply="all" type=SGML>

<LEX type="PHRASE"
     file_name="&NUMEX-DIR;/timex.lex"
     alias="TIM_lex"  ></LEX>

<!-- Monday Tuesday ...  -->
  <RULE  name="day-name-full" targ_sg='TIMEX TYPE DATE' >
    <REL match="&CAPWRD;" >
      <CONSTR  check_in="TIM_lex" check_tags="DY *"></CONSTR>
    </REL>
  </RULE>

<!-- Mon. Mon Tues. Tues ...  -->
  <RULE  name="day-name-abbrev" targ_sg='TIMEX TYPE DATE'>
    <REL match="&CAPWRD;" >
      <CONSTR  check_in="TIM_lex" check_tags="DYA *"></CONSTR>
    </REL>
    <REL  match="W[C="."]" m_mod=QUEST></REL>
  </RULE>

<!-- Monday Mon. Mon   ...  -->
  <RULE  name="day-name"  type=DISJF >
    <REL    type=REF  match="day-name-full"></REL>
    <REL    type=REF  match="day-name-abbrev"></REL>
  </RULE>
</RULES>
```

Figure 2: Examples of SGML transduction rules for recognising certain temporal expressions

files was done by a number of different people, working within XML with commonly agreed DTDs was found to be helpful.

The `TIMEX` and `NUMEX` components of our MUC system do not make use of part-of-speech tagged information, and can be run before or after `lttok` and `ltstop`.

4 ENAMEX

For recognising `enamex` elements, we similarly compiled grammars and collected resources, such as names of locations and organisations, first names (for use in name recognition), etc. But as demonstrated in section 1, a MUC system cannot rely too much on such lists, and different strategies need to be used for high-precision `enamex` recognition. In fact, we have

also run our NE system without any lexical resources and report on these experiments in [9].

The basic philosophy underlying our approach is as follows. When looking at a string of words like "Adam Kluver" it is not possible to say whether this is the name of a person or an organisation. However, somewhere in the text, there is likely to be some contextual material which makes it clear which of those it is. Our strategy is to only make a decision once we have identified this bit of contextual information.

We further assume that, once we have identified contextual material which makes it clear that "Adam Kluver" is (e.g.) the name of a company, then any other mention of "Adam Kluver" in that document will be referring to that company. If the author at some point had wanted to also refer to a person called "Adam Kluver", s/he would have provided some extra context to make this clear, and this context would have been picked up in the first step.

If no suitable context is found anywhere in the text to decide what sort of Named Entity "Adam Kluver" is, the system can check other resources, e.g. a list of known company names. But this method only applies after substantial context checking has been carried out.

In our MUC system, we implemented this approach as a combination of symbolic transduction of SGML elements with probabilistic partial matching, in 5 stages:

1. sure-fire rules
2. partial match 1
3. relaxed rules
4. partial match 2
5. title assignment

We describe each in turn.

ENAMEX: 1. Sure-fire Rules

In the first step, our SGML transducer `fsgmatch` is used with sure-fire rules. These rules are very context-oriented and they fire only when a possible candidate expression is surrounded by a suggestive context. Sure-fire rules rely on known corporate designators (Ltd., Inc., etc.), person titles (Mr., Dr., Sen.), and definite contexts such as those in Figure 3. The sure-fire rules apply after POS tagging, so at this stage words like "analyst" have already been identified as `PROF` (professions), and words like "brother" as `REL` (relatives).

An example of a transduction rule is presented in Figure 4.

At this stage our MUC system treats information from the lists as *likely* rather than definite and always checks if the context is either suggestive or non-contradictive. For example, a likely company name with a conjunction is left untagged at this stage if the company is not listed in a list of known companies: in a sentence like "this was good news for China International Trust and Investment Corp", it is not clear whether the text deals with one or two companies, and no markup is applied.

Context Rule	Assign	Example
Xxxx+ is? a? JJ* PROF	PERS	Yuri Gromov, a former director
Xxxx+ is? a? JJ* REL	PERS	John White is beloved brother
Xxxx+ himself	PERS	White himself
Xxxx+, DD+,	PERS	White, 33,
shares in Xxxx+	ORG	shares in Trinity Motors
PROF of/at/with Xxxx+	ORG	director of Trinity Motors
in/at LOC	LOC	in Washington
Xxxx+ area	LOC	Beribidjan area

Figure 3: Examples of sure-fire transduction material for **enamex**. Xxxx+ is a sequence of capitalized words; DD is a digit; PROF is a profession; REL is a relative; JJ* is a sequence of zero or more adjectives; LOC is a known location.

Similarly, the system postpones the markup of unknown organizations whose name starts with a sentence initial common word, as in "Suspended Ceiling Contractors Ltd denied the charge". Since the sentence-initial word has a capital letter, it could be an adjective modifying the company "Ceiling Contractors Ltd", or it could be part of the company name, "Suspended Ceiling Contractors Ltd".

Names of possible locations found in our gazetteer of place names are marked as `LOCATION` only if they appear with a context that is suggestive of location. "Washington", for example, can just as easily be a surname or the name of an organization. Only in a suggestive context, like "in the Washington area", will it be marked up as location.

ENAMEX: 2. Partial Match 1

After the sure-fire symbolic transduction the system performs a probabilistic partial match of the identified entities. This is implemented as an interaction between two tools. The first tool collects all named entities already identified in the document. It then generates all possible partial orders of the composing words preserving their order, and marks them if found elsewhere in the text. In our example, "Adam Kluver Ltd" had already been recognised as an organisation by the sure-fire rule. In this second step, any occurrences of "Adam Kluver", "Kluver Ltd", "Adam Ltd" and "Adam Kluver" are also tagged as *possible* organizations. This markup, however, is not definite since some of these words (such as "Adam") could refer to a different entity.

This annotated stream goes to a second tool, a pre-trained maximum entropy model. It takes into account contextual information for named entities, such as their position in the sentence, whether they exist in lowercase in general, whether they were used in lowercase elsewhere in the same document, etc. These features are passed to the model as attributes of the partially matched words. If the model provides a positive answer for a partial match, the match is wrapped into a corresponding `ENAMEX` element.

```
<RULE name="FirstNameDictExact" type=DISJ >
   <REL  match = "W/#~[A-Z][a-z]+$" >
       <LEXCHECK  check_in= "NAMES_lex" check_tags="FN *" ></LEXCHECK>
   </REL>
   <REL  match = "W/#~[A-Z][a-z]+((slav)|(ldo))$" >
</RULE>

<RULE name="John_Uuuu+_Xxxx_Jr?"  type=SEQ  targ='ENAMEX TYPE=PERSON'  >
   <REL match = "FirstNameDictExact"      type="REF"                  ></REL>
   <REL match = "W[C=NNP]"                                m_mod= STAR   ></REL>
   <REL match = "W[C=NNP L!=d]"                           m_mod= PLUS   ></REL>
   <REL match = 'W/#~^((Jr\.?)|(Sr\.?))'                  m_mod= QUEST  ></REL>
</RULE>
```

Figure 4: Sample of a transduction rule. The rule `John_Uuuu+_Xxxx_Jr?` calls subrules in sequence by referencing the rule `FirstNameDictExact`.

ENAMEX: 3. Rule Relaxation

Once this has been done, the system again applies the SGML transduction rules. But this time the rules have much more relaxed contextual constraints and extensively use the information from already existing markup and from the lexicon compiled during processing, e.g. containing partial orders of already identified named entities.

At this stage the system will mark word sequences which look like person names. For this it uses a grammar of names: if the first capitalized word occurs in a list of first names and the following word(s) are unknown capitalized words, then this string can be tagged as a `PERSON`. Here we are no longer concerned that a person name can refer to a company. If the name grammar had applied earlier in the process, it might erroneously have tagged "Adam Kluver" as a `PERSON` instead of an `ORGANIZATION`. At this point in the chain of **`enamex`** processing, that is not a problem anymore: "Adam Kluver" will by now already have been identified as an `ORGANIZATION` by the sure-fire rules or during partial matching. If it hasn't, then it is likely to be the name of a person.

At this stage the system will also attempt to resolve conjunction problems in names of organisations. For example, in "this was good news for China International Trust and Investment Corp", it is not clear whether the text is referring to one organisation or two. The system checks if possible parts of the conjunctions were used in the text on their own and thus are names of different organizations; if not, the system has no reason to assume that more than one company is being talked about.

In a similar vein, the system resolves the attachment of sentence initial capitalized modifiers, the problem alluded to above with the "Suspended Ceiling Contractors Ltd" example: if the modifier was seen with the organization name elsewhere in the text, then the system has good evidence that the modifier is part of the company name; if the modifier does not occur

anywhere else in the text with the company name, it is assumed not to be part of the it.

At this stage known organizations and locations from the lists available to the system are marked in the text, again without checking the context in which they occur.

ENAMEX: 4. Partial Match 2

At this point, the system has exhausted its resources (name grammar, list of locations, etc). The system then performs another partial match to annotate names like "White" when "James White" had already been recognised as a person, and to annotate company names like "Hughes" when "Hughes Communications Ltd." had already been identified as an organisation. As in Partial Match 1, this process of partial matching is again followed by a probabilistic assignment supported by the maximum entropy model.

ENAMEX: 5. Title Assignment

Because titles of news wires are in capital letters, they provide little guidance for the recognition of names. In the final stage of `enamex` processing, entities in the title are marked up, by matching or partially matching the entities found in the text, and checking against a maximum-entropy model trained on document titles. For example, in "GENERAL TRENDS ANALYST PREDICTS LITTLE SPRING EXPLOSION" "GENERAL TRENDS" will be tagged as an organization because it partially matches "General Trends Inc" elsewhere in the text, and "LITTLE SPRING" will be tagged as a location because elsewhere in the text there is supporting evidence for this hypothesis.

5 Conclusion

5.1 Performance

In the MUC competition, our system's combined precision and recall score was 93.39%. This was the highest score, better in a statistically significant way than the score of the next best system. Scores varied from 93.39% to 69.67%. Further details on this can be found in [8].

The table in Figure 5 shows the progress of the performance of the system we fielded for the MUC competition through the five stages.

As one would expect, the sure-fire rules give very high precision (around 96-98%), but very low recall—in other words, they don't find many `enamex` entities, but the ones they find are correct. Subsequent phases of processing add gradually more and more `enamex` entities (recall increases from around 40% to around 90%), but on occasion introduce errors (resulting in a slight drop in precision). Our final score for `ORGANISATION`, `PERSON` and `LOCATION` is given in the bottom line of Figure 5.

Stage	ORGANIZATION		PERSON		LOCATION	
Sure-fire Rules	R: 42	P: 98	R: 40	P: 99	R: 36	P: 96
Partial Match 1	R: 75	P: 98	R: 80	P: 99	R: 69	P: 93
Relaxed Rules	R: 83	P: 96	R: 90	P: 98	R: 86	P: 93
Partial Match 2	R: 85	P: 96	R: 93	P: 97	R: 88	P: 93
Title Assignment	R: 91	P: 95	R: 95	P: 97	R: 95	P: 93

Figure 5: Scores obtained by the system through different stages of the analysis. R - recall P - precision.

5.2 The system

One of the design features of the system which sets it apart from other Named Entity recognition systems is that it is designed fully within the SGML paradigm: the system is composed of several tools which are connected via a pipeline with data encoded in SGML or XML. This allows the same tool to apply different strategies to different parts of the texts using different resources. The tools do not convert from SGML into an internal format and back, but operate at the SGML or XML level.

Our system does not rely heavily on lists or gazetteers but instead treats information from such lists as "likely" and concentrates on finding contexts in which such likely expressions are definite. In fact, the first phase of the `enamex` analysis uses virtually no lists but still achieves substantial recall.

The system is document centred. This means that at each stage the system makes decisions according to a confidence level that is specific to that processing stage, and draws on information from other parts of the document. The system is hybrid, applying symbolic rules and statistical partial matching techniques in an interleaved fashion.

5.3 Limitations

Unsurprisingly, the major problem for the system are single capitalized words, mentioned just once or twice in the text and without suggestive contexts. In such a case the system cannot apply contextual assignment, assignment by analogy or lexical lookup and fails to markup the entity. As the results of the MUC competition show, this is a relatively rare occurrence.

6 Availability

A runtime version of the system described here is available for free at
`http://www.ltg.ed.ac.uk/software/ne/`.

We also have a set of tools which can be used to develop a Named Entity recognition system. The tool suite is called LT TTT, and is available from
`http://www.ltg.ed.ac.uk/software/ttt/`.

LT TTT consists of `lttok`, `ltstop` and `fsgmatch`, a number of resource files for tokenisation, for end-of-sentence disambiguation, and for the recognition of temporal expressions, and tools for extending these resource grammars or for creating new ones.

It also has a visual interface which uses XSL style sheets to render the XML Named Entity annotation in a form that is easier to inspect.

The part of speech tagger is available as a separate tool. See `http://www.ltg.ed.ac.uk/software/pos/`.

Acknowledgements

The work reported in this paper was supported in part by grant GR/L21952 (Text Tokenisation Tool) from the Engineering and Physical Sciences Research Council, UK. For help with the development of the MUC system authors wish to thank Colin Matheson, Steven Finch and Irina Nazarova. Henry Thompson, David McKelvie, Richard Tobin and many other members of the Language Technology Group contributed to the development of the many LTG tools that were used in the development of the MUC system.

References

[1] C. Brew and D. McKelvie 1996. "Word Pair Extraction for Lexicography." In *Proceedings of NeM-LaP'96*, pp 44–55. Ankara, Turkey.

[2] S. Finch and M. Moens 1995. "SISTA final report." Available from: Language Technology Group, University of Edinburgh.

[3] A. Mikheev and S. Finch 1995. "Towards a Workbench for Acquisition of Domain Knowledge from Natural Language." In *Proceedings of the 7th Conference of the European Chapter of the Association for Computational Linguistics (EACL'95)*, pp 194–201. Dublin.

[4] A. Mikheev and S. Finch 1997. "A Workbench for Finding Structure in Texts" In *Proceedings of the Fifth Conference on Applied Natural Language Processing*, pp 372–379. Washington D.C.

[5] A. Mikheev. 1997 "Automatic Rule Induction for Unknown Word Guessing." In *Computational Linguistics* **23** (3), pp 405–423.

[6] A. Mikheev. 1997 "LT POS – the LTG part of speech tagger." Language Technology Group, University of Edinburgh. `http://www.ltg.ed.ac.uk/software/pos`

[7] A. Mikheev. 1998 "Feature Lattices for Maximum Entropy Modelling" In *Proceedings of the 36th Conference of the Association for Computational Linguistics*, pp 848–854. Montreal, Quebec.

[8] A. Mikheev, C. Grover and M. Moens 1998 Description of the LTG system used for MUC-7. In *Seventh Message Understanding Conference (MUC–7): Proceedings of a*

Conference held in Fairfax, VA, April 29–May 1, 1998. Los Altos: Morgan Kaufmann, forthcoming.

[9] A. Mikheev, C. Grover and M. Moens 1998 Recognising Named Entities without Lists. Unpublished. Edinburgh, Language Technology Group.

[10] D. McKelvie, C. Brew and H.S. Thompson 1998. "Using SGML as a Basis for Data-Intensive Natural Language Processing." In *Computers and the Humanities* **35**, pp 367–388.

[11] M. Spiegel 1985 "Pronouncing surnames automatically" *Proceedings of the American Voice Input/Ouput Society*

[12] H.S. Thompson, D. McKelvie and S. Finch 1997. "The Normalised SGML Library LT NSL version 1.5." Language Technology Group, University of Edinburgh. `http://www.ltg.ed.ac.uk/software/nsl`

[13] H.S. Thompson, C. Brew, D. McKelvie, A. Mikheev and R. Tobin 1998. "The XML Library LT XML version 1.0." Language Technology Group, University of Edinburgh. `http://www.ltg.ed.ac.uk/software/xml`