D1.4.1a Language Issues

Diana Maynard (University of Sheffield)

with contributions from:
Luis Rodrigo Aquado (ISOCO)
Fang Huang (University of Sheffield)

Abstract.
EU-IST Integrated Project (IP) IST-2003-506826 SEKT
Deliverable D1.4.1(WP1)

The aim of task T1.4 is to develop automatic methods for dealing with characteristics of a particular natural language. This includes automatic construction of target language vocabularies and thesauri, lemmatization and stemming procedures, and models for dealing with multilingual aspects of language processing.
This deliverable describes some multilingual processing components developed within GATE (an architecture for language processing). Three applications, or sets of components, have been developed, for German, French and Spanish respectively. The applications consist of a conditional controller operating over a pipeline of processing resources, which run over the language resources (a corpus of documents). This report describes a user guide to running the applications, and gives details of the components, evaluations carried out, some related work and future plans.
Keyword list: information extraction, language processing, language resources

Document Id.	SEKT/2004/D1.4.1a/v1.0
Project	SEKT EU-IST-2003-506826
Date	Dec 13, 2004
Distribution	public

SEKT Consortium

This document is part of a research project partially funded by the IST Programme of the Commission of the European Communities as project number IST-2003-506826.

British Telecommunications plc.
Orion 5/12, Adastral Park
Ipswich IP5 3RE
UK
Tel: +44 1473 609583, Fax: +44 1473 609832
Contact person: John Davies
E-mail: john.nj.davies@bt.com

Jozef Stefan Institute
Jamova 39
1000 Ljubljana
Slovenia
Tel: +386 1 4773 778, Fax: +386 1 4251 038
Contact person: Marko Grobelnik
E-mail: marko.grobelnik@ijs.si

University of Sheffield
Department of Computer Science
Regent Court, 211 Portobello St.
Sheffield S1 4DP
UK
Tel: +44 114 222 1891, Fax: +44 114 222 1810
Contact person: Hamish Cunningham
E-mail: hamish@dcs.shef.ac.uk

Intelligent Software Components S.A.
Pedro de Valdivia, 10
28006 Madrid
Spain
Tel: +34 913 349 797, Fax: +49 34 913 349 799
Contact person: Richard Benjamins
E-mail: rbenjamins@isoco.com

Ontoprise GmbH
Amalienbadstr. 36
76227 Karlsruhe
Germany
Tel: +49 721 50980912, Fax: +49 721 50980911
Contact person: Hans-Peter Schnurr
E-mail: schnurr@ontoprise.de

Vrije Universiteit Amsterdam (VUA)
Department of Computer Sciences
De Boelelaan 1081a
1081 HV Amsterdam
The Netherlands
Tel: +31 20 444 7731, Fax: +31 84 221 4294
Contact person: Frank van Harmelen
E-mail: frank.van.harmelen@cs.vu.nl

Empolis GmbH
Europaallee 10
67657 Kaiserslautern
Germany
Tel: +49 631 303 5540, Fax: +49 631 303 5507
Contact person: Ralph Traphöner
E-mail: ralph.traphoener@empolis.com

University of Karlsruhe, Institute AIFB
Englerstr. 28
D-76128 Karlsruhe
Germany
Tel: +49 721 608 6592, Fax: +49 721 608 6580
Contact person: York Sure
E-mail: sure@aifb.uni-karlsruhe.de

University of Innsbruck
Institute of Computer Science
Techikerstraße 13
6020 Innsbruck
Austria
Tel: +43 512 507 6475, Fax: +43 512 507 9872
Contact person: Jos de Bruijn
E-mail: jos.de-bruijn@deri.ie

Kea-pro GmbH
Tal
6464 Springen
Switzerland
Tel: +41 41 879 00, Fax: 41 41 879 00 13
Contact person: Tom Bösser
E-mail: tb@keapro.net

Sirma AI EAD, Ontotext Lab
135 Tsarigradsko Shose
Sofia 1784
Bulgaria
Tel: +359 2 9768 303, Fax: +359 2 9768 311
Contact person: Atanas Kiryakov
E-mail: naso@sirma.bg

Universitat Autonoma de Barcelona
Edifici B, Campus de la UAB
08193 Bellaterra (Cerdanyola del Vallès)
Barcelona
Spain
Tel: +34 93 581 22 35, Fax: +34 93 581 29 88
Contact person: Pompeu Casanovas Romeu
E-mail: pompeu.casanovas@uab.es

Executive Summary

This deliverable describes some multilingual processing components developed within GATE (an architecture for language processing). Three applications, or sets of components, have been developed, for German, French and Spanish respectively. The applications consist of a conditional controller operating over a pipeline of processing resources, which run over the language resources (a corpus of documents). The first two sections of this report provide a general introduction to the task and to setting up and running GATE and the applications. The next 3 sections detail the respective applications. Section 6 describes some stemming processing resources developed in GATE for a variety of different languages. Section 7 describes some initial evaluations carried out on the German and French applications. Section 8 discusses some related work on adapting IE applications to other languages, and finally Section 9 outlines some proposed future improvements to the current work.

Contents

1 Introduction

The aim of task T1.4 is to develop automatic methods for dealing with characteristics of a particular natural language. This includes automatic construction of target language vocabularies and thesauri, lemmatization and stemming procedures, and models for dealing with multilingual aspects of language processing.

This deliverable describes some multilingual processing components developed within GATE. GATE is an architecture for language processing developed at the University of Sheffield [3], and is freely available for research purposes. It contains a default Information Extraction (IE) system for English, ANNIE, which consists of a pipeline operating over a set of Processing Resources (PRs). Some previous work has been carried out to adapt this IE system to new languages and domains [8, 7, 9] (Chinese, Arabic, Russian, Hindi, Cebuano), but there has been little previous work on producing multilingual components in GATE for the more common European languages, such as French, German, Spanish and Italian. Such components are the cornerstone for many semantic web and other knowledge-intensive applications, either as full IE systems or as individual components (POS taggers, stemmers, keyword extractors etc.).

Three applications, or sets of components, have been developed, for German, French and Spanish respectively. The applications consist of a conditional controller operating over a pipeline of processing resources, which run over the language resources (a corpus of documents). The next section of this report provides a brief user guide to installing and running the applications. The next 3 sections detail the respective applications. Section 6 describes some stemming processing resources developed in GATE for a variety of different languages. Section 7 describes some initial evaluations carried out on the German and French applications. Section 8 discusses some related work on adapting IE applications to other languages, and finally Section 9 outlines some proposed future improvements to the current work.

2 User guide

This section details the requirements needed, and explains how to download, install and run the relevant tools and applications.

To set up the application for the first time:

1. Download and install GATE 3 from http://gate.ac.uk (note that the application will not run on earlier versions of GATE);

2. If you want to use the French or German applications, download and install the TreeTagger from http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/DecisionTreeTagger.html, following the instructions given. Note that it will only work on Linux unless a special licence is obtained for the Windows version (which is not free).

3. Start up GATE (see the User Guide for more information, as the method may depend on

your installation). Note that GATE can be run on any platform with no problems from the TreeTagger – it is simply that the TreeTagger must be installed on Linux.

4. select ”Restore Application from File” and load the relevant application for the language. The German application can be found in gate/plugins/german/german.gapp; the French application can be found in gate/plugins/french/french.gapp and the Spanish application can be found in gate/plugins/spanish/spanish.gapp. Note that the version of the Spanish application containing the keyword annotator can only be found in the deliverable accompanying this report, as it is not publically available.

5. For French and German, you will need to add the TreeTagger to the application yourself for the first time.

   (a) Right-click on Processing Resources and select New –> TreeTagger.

   (b) Select OK (give the Tagger a name if you wish, or leave it blank).

   (c) Double-click on the name of the application to take you to the Application Editor.

   (d) Here you must add the TreeTagger to the set of Selected Processing Resources, using the arrow. Move the TreeTagger up the list of Selected Resources until it lies between the Splitter and the Gazetteer.

   (e) Select the TreeTagger and you will see a set of parameters for it below.

   (f) Type NE as the value for AnnotationSetName

   (g) Select the binary for the TreeTagger using the Browse icon: the file to select should be in the directory cmd/tree-tagger-german or cmd/tree-tagger-french wherever you have installed the TreeTagger.

6. Finally, save the new application you have created so that you do not have to repeat this process every time, by right clicking on the name of the application and selecting Save Application State. You can overwrite the original application or save it somewhere new. For Spanish you do not need to save the application, as you will not have modified it.

To run the application in GATE and view the results, the following steps should be undertaken:

1. start up GATE;

2. select ”Restore Application from File” and load the relevant application for your chosen language;

3. load a corpus of texts for your chosen language (there are some sample texts in the data directories for each language plugin);

4. add the corpus to the application using the Application Viewer;

5. select ”Run” from the Application Viewer;

6. view the results by selecting a document from the corpus;

7. the annotations produced will be in an annotation set named ”NE”, viewable by opening the AnnotationSets viewer.

For more detailed descriptions of installing and running GATE, see the GATE User Guide [4].

In the following sections, we describe the individual processing resources for each of the three languages in more detail.

3 German language processing components

3.1 Architecture

The German application consists of the following processing resources:

- multilingual tokeniser
- German sentence splitter
- German part-of-speech (POS) tagger
- English gazetteer
- German gazetteer
- German semantic tagger
- multilingual coreferencer

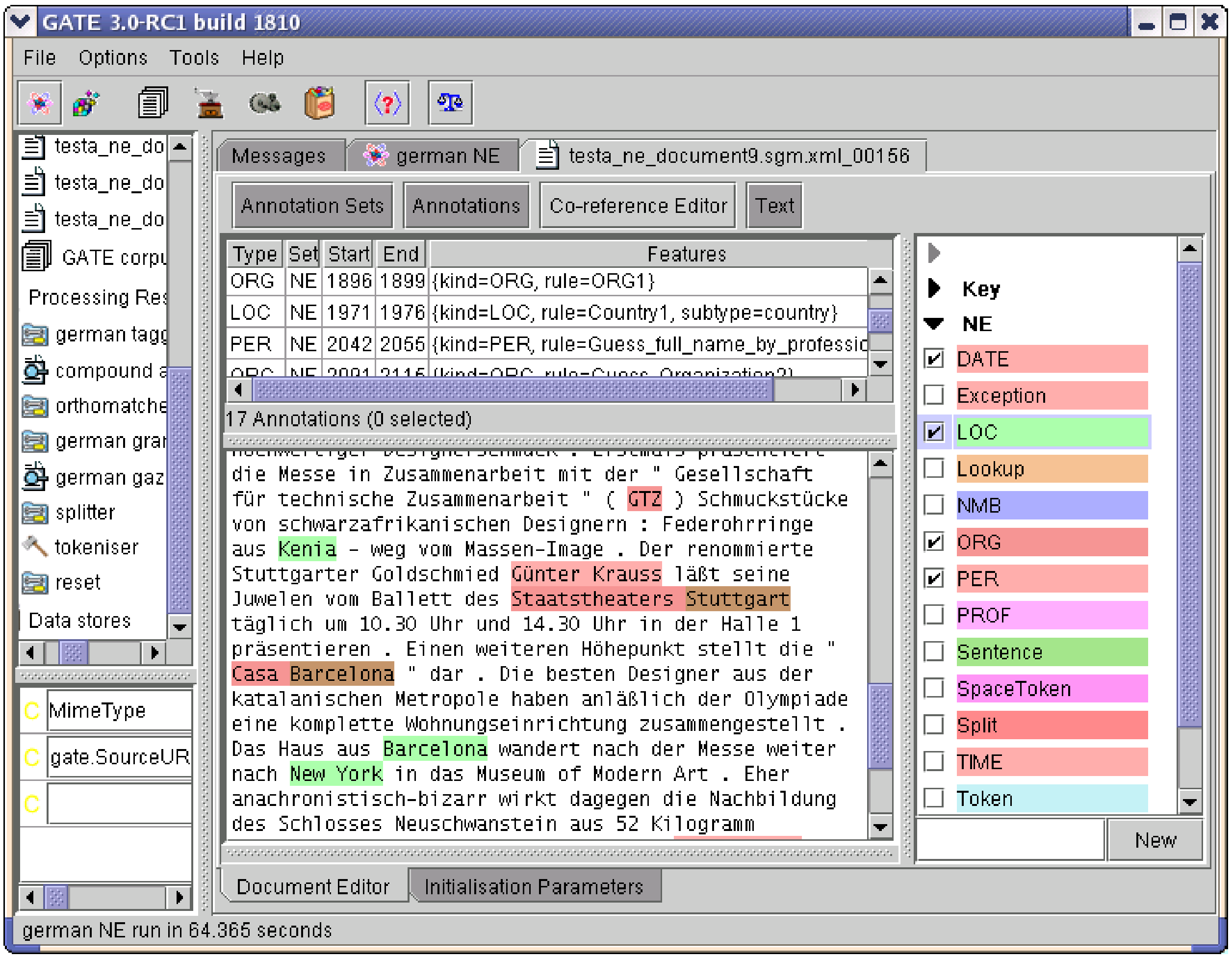
The application reuses some previous work (mainly gazetteer lists and some grammar rules) carried out in the context of the DotKOM project (EU IST Framework V grant no. IST-2001-34038).

Figure 1 shows a German text in GATE annotated with various annotation types. In the following sections, we describe each of the processing resources in more detail.

3.2 Tokeniser

The tokeniser splits the text into very simple tokens such as numbers, punctuation and words of different types. For example, we distinguish between words in uppercase and lowercase, and between certain types of punctuation. The aim is to limit the work of the tokeniser to maximise efficiency, and enable greater flexibility by placing the burden on the grammar rules, which are more adaptable.

A **word** is defined as any set of contiguous upper or lowercase letters, including a hyphen (but no other forms of punctuation). A word also has the attribute “orth”, for which four values are defined:



Type	Set	Start	End	Features
ORG	NE	1896	1899	{kind=ORG, rule=ORG1}
LOC	NE	1971	1976	{kind=LOC, rule=Country1, subtype=country}
PER	NE	2042	2055	{kind=PER, rule=Guess_full_name_by_professio
ORG	NE	2091	2116	{kind=ORG, rule=Guess_Organization2}

17 Annotations (0 selected)

```
die Messe in Zusammenarbeit mit der " Gesellschaft
für technische Zusammenarbeit " ( GTZ ) Schmuckstücke
von schwarzafrikanischen Designern : Federohrringe
aus Kenia - weg vom Massen-Image . Der renommierte
Stuttgarter Goldschmied Günter Krauss läßt seine
Juwelen vom Ballett des Staatstheaters Stuttgart
täglich um 10.30 Uhr und 14.30 Uhr in der Halle 1
präsentieren . Einen weiteren Höhepunkt stellt die "
Casa Barcelona " dar . Die besten Designer aus der
katalanischen Metropole haben anläßlich der Olympiade
eine komplette Wohnungseinrichtung zusammengestellt .
Das Haus aus Barcelona wandert nach der Messe weiter
nach New York in das Museum of Modern Art . Eher
anachronistisch-bizarr wirkt dagegen die Nachbildung
des Schlosses Neuschwanstein aus 52 Kilogramm
```

[✓] DATE
[ ] Exception
[✓] LOC
[ ] Lookup
[ ] NMB
[✓] ORG
[✓] PER
[ ] PROF
[ ] Sentence
[ ] SpaceToken
[ ] Split
[ ] TIME
[ ] Token

german NE run in 64.365 seconds

Figure 1: Annotated German text in GATE

- upperInitial - initial letter is uppercase, rest are lowercase
- allCaps - all uppercase letters
- lowerCase - all lowercase letters
- mixedCaps - any mixture of upper and lowercase letters not included in the above categories

A **number** is defined as any combination of consecutive digits. There are no subdivisions of numbers. Numbers containing other characters such as punctuation are not identified as such at this stage, e.g. ”32.6” is annotated as a sequence of 3 tokens: number, punctuation and number. It is the work of the semantic tagger to combine these three tokens into a number annotation, if required. The reason for this is to maximise flexibility and efficiency, as explained earlier.

Two types of **symbol** are defined: currency symbol (e.g. ‘$’, ‘£’) and symbol (e.g. ‘&’, ‘ˆ’). These are represented by any number of consecutive currency or other symbols (respectively).

Three types of **punctuation** are defined: start_punctuation (e.g. ‘(’), end_punctuation (e.g. ‘)’), and other punctuation (e.g. ‘:’). Each punctuation symbol is a separate token.

White spaces are divided into two types of **SpaceToken** - space and control - according to whether they are pure space characters or control characters. Any contiguous (and homogenous) set of space or control characters is defined as a SpaceToken.

A tokeniser rule has a left hand side (LHS) and a right hand side (RHS). The LHS is a regular expression which has to be matched on the input; the RHS describes the annotations to be added to the AnnotationSet. The LHS is separated from the RHS by ’>’. The following operators can be used on the LHS:

```
| (or)
* (0 or more occurrences)
? (0 or 1 occurrences)
+ (1 or more occurrences)
```

The RHS uses ’;’ as a separator, and has the following format:

```
{LHS} > {Annotation type};{attribute1}={value1};...;{attribute
n}={value n}
```

The following tokeniser rule is for a word beginning with a single capital letter:

```
"UPPERCASE_LETTER" "LOWERCASE_LETTER"* >
  Token;orth=upperInitial;kind=word;
```

It states that the sequence must begin with an uppercase letter, followed by zero or more lowercase letters. This sequence will then be annotated as type “Token”. The attribute “orth” (orthography) has the value “upperInitial”; the attribute “kind” has the value “word”.

3.3 Sentence Splitter

The sentence splitter is a cascade of finite-state transducers which segments the text into sentences. This module is required for the tagger. Each sentence is annotated with the type Sentence. Each sentence break (such as a full stop) is also given a “Split” annotation. This has several possible types: “.”, “punctuation”, “CR” (a line break) or “multi” (a series of punctuation marks such as “?!?!”).

The splitter has been adapted from the English sentence splitter that is available within GATE. It differs from the English one in that it uses a list of common German abbreviations in order to prevent the erroneous termination of a sentence after such an abbrevation, where the full stop does not denote the sentence end but is part of the abbreviation.

3.4 German POS tagger

The German POS tagger PR consists of a wrapper for the TreeTagger, which is a language-independent part-of-speech tagger developed by the University of Stuttgart. The TreeTagger currently supports English, French, German, and Italian. Only the Linux version is freely available, however, so our system will only run on Linux and not on Windows. We plan to include a future version of the system for Windows which does not make use of the tagger, or which uses an alternative tagger (see Section 9). The TreeTagger has been integrated with GATE using a GATE CREOLE wrapper, originally designed by the CLaC lab (Computational Linguistics at Concordia), Concordia University, Montreal (http://www.cs.concordia.ca/CLAC). The GATE wrapper calls the TreeTagger as an external program, passing GATE documents as input, and generates new annotations of type ”TreeTaggerToken”, with the features length, string and category (which contains the POS generated by the TreeTagger). The reason the POS features are not simply added to the existing Token annotations produced by the tokeniser, as the English POS tagger in GATE does, is because the Tokens produced by the tokeniser are not in all cases the same as those expected by the TreeTagger, so this can lead to problems. It is therefore better to generate a new annotation. Some further descriptions of the TreeTagger and its use can be found in the GATE User Guide [4].

The POS tags generated by the TreeTagger for German are extremely complicated, so we do not describe them here. A full description of the tags can be found at http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/DecisionTreeTagger.html. The main tags used are as follows:

N	Noun
V	Verb
ART	Article
ADJ	Adjective
P	Pronoun
CARD	Cardinal Number
ADV	Adverb
KO	Conjunction
AP	Adposition

ITJ	Interjection
PTK	Particle

3.5 English Gazetteer

The gazetteer consists of a set of lists, which are plain text files, with one entry per line. Each list represents a set of names, such as names of cities, organisations, days of the week, etc. Gazetteer lists can be set at runtime to be either case sensitive or case insensitive. Here we use them in case sensitive mode, which means that we need to specify alternatives if for example the first letter of a word is likely to appear either capitalised or in lower case. The advantage of using case-sensitive mode is that it avoids many problems of ambiguity between common words and the proper nouns which we wish to identify in the lists.

Below is a small section of the list for units of currency:

```
Ecu
European Currency Units
FFr
Fr
German mark
German marks
New Taiwan dollar
New Taiwan dollars
NT dollar
NT dollars
```

An index file (lists.def) is used to access the lists; for each list, a major type is specified and, optionally, a minor type and a language. The major and minor type are simply keys by which we can refer to annotations found, for example, we might give a list of mountains the major type ”country” and the minor type ”mountain”. In the example below, the first column (where columns are separated by semicolons) refers to the list name, the second column to the major type, the third to the minor type, and the fourth to the language.

```
currency_prefix.lst:currency_unit:pre_amount:english
currency_unit.lst:currency_unit:post_amount:english
date.lst:date:specific:english
day.lst:date:day:english
```

These lists are compiled into finite state machines. Any text matched by these machines will be annotated with features specifying the major and minor types,and the language. So, for example, if a specific day needs to be identified, the minor type “day” could be specified in the grammar, in order to match only information about specific days; if any kind of date needs to be identified, the major type “date” could be specified, to enable text annotated with any information about dates to be identified.

3.6 German gazetteer

The German gazetteer is like the English gazetteer described in Section 3.5, but contains elements specific to German. Not all English lists need translating into (or equivalent finding in) German: for example, many names of people, locations etc remain the same in both languages. However, there are also many words such as times, dates, numbers, and keywords, which must be provided in German. All German lists have the feature ”language” with value ”German”, so that they can be identified as such if necessary.

3.7 German semantic tagger

The German semantic tagger was adapted from the English semantic tagger available with GATE. The semantic tagger consists of a set of grammars based on the JAPE language [5]. They contain rules which act on annotations assigned previously, in order to produce outputs of annotated entities. Each grammar set contains a series of JAPE grammars run sequentially, such that annotations created by one grammar may be used by a following grammar. This is very important for ambiguity resolution between entity types.

Each JAPE grammar, or phase, consists of a set of pattern/action rules. The phases run sequentially and constitute a cascade of finite state transducers over annotations. The left-hand-side (LHS) of the rules consists of an annotation pattern that may contain regular expression operators (e.g. `*, ?, +`). The right-hand-side (RHS) consists of annotation manipulation statements. Annotations matched on the LHS of a rule may be referred to on the RHS by means of labels that are attached to pattern elements.

At the beginning of each grammar, two options can be set:

- Control - this defines the method of rule matching. This can be one of three types: brill, appelt or first. The Brill style means rules fire on all possible matches; Appelt style involves a set of priorities (longest first, then explicit priority defined in each rule); first style means that the shortest rule fires.
- Debug - when set to true, if the grammar is running in Appelt style and there is more than one possible match, the conflicts will be displayed on the standard output.

Input annotations must also be defined at the start of each grammar. These are the annotations that will be matched by the rules: any annotation types not defined will be ignored in the matching process. If no annotations are defined, the default will be Token, SpaceToken and Lookup (i.e. only these annotations will be considered when attempting a match).

Patterns are specified in terms of annotations previously assigned to text and may optionally include features and values associated with the annotations. Any kind of annotation may be used - typically, this will be either a text string, an annotation from a previous processing resource such as the result of gazetteer lookup or POS tagging, or an annotation created by a previous grammar phase.

The same operators can be used for the pattern matching as for the tokeniser rules, i.e.

```
| (or)
* (0 or more occurrences)
? (0 or 1 occurrences)
+ (1 or more occurrences)
```

The RHS of the rule, in its simple form, contains information about the annotation to be created (its type, and any features and values it may contain). In its more complex form, it can also contain any arbitrary Java code, which is frequently used to perform more complex operations over annotations, such as percolating features or deleting temporary annotations.

In the simple example below, the pattern described will be awarded an annotation of type "Organization". The pattern consists of one or more Tokens beginning with an uppercase letter, followed by something that is contained in a gazetteer list of cities. The rule guesses that this pattern is likely to indicate an organisation.

```
Rule: Guess_Organization2
(
    ({Token.orth == upperInitial})+
    {Lookup.minorType == "city"}
):OrgName3
-->
:OrgName3.Organization =
        {kind = "ORG", rule = "Guess_Organization2"}
```

The German semantic tagger currently consists of the following 8 phases:

- compound_analysis
- time_date
- profession
- person
- location
- org
- number
- money

These phases each find annotations belonging to the respective annotation type, e.g. the Person phase finds Person annotations. The Profession phase finds annotations corresponding to people's professions: although these are not a required annotation type to be identified, they are useful in aiding with recognition of other annotation types such as Person.

One of the major differences between the German semantic tagger and the GATE English semantic tagger (on which it is loosely based) is the presence of the compound analysis. This is because, unlike English, German is a compounding language, where two or more nouns are often joined together to form longer compounds. Whereas in English these would be separate tokens, and keywords can often be used to identify entities (for example, ”North” preceding a proper noun often indicates a Location), in German this is more difficult because the tokens have to be broken down into their individual components in order to make use of this kind of analysis. We therefore have a grammar phase called compound_analysis, which uses a special gazetteer to identify such keywords commonly found as part of longer words. The grammar rules then produce temporary annotations over such words, which are used in later phases of the grammar.

For example, in English we might have a grammar rule which looks for a proper noun followed by the word ”city”. This pattern might indicate the presence of a Location. For German, we might have a word like ”Darmstadt”, which essentially consists of the compound ”Darm” + ”Stadt” (town). If we do not have information that Darmstadt is a city in our gazetteer list, but we know that ”stadt” is a typical location keyword, we can use a pattern such as the following, where ”stadt” will have been identified by the gazetteer as having the type ”location_key”. The following rule looks for a pattern of one or more words beginning with an uppercase letter, an optional hyphen, and then a location keyword, and annotates the whole pattern as a Location.

```
Rule: Location_key2
(
  (
    (
     {Token.orth == "allCaps"}|
     {Token.orth == "upperInitial"}
    )
    ({Token.string == "\-"})
   )*
   {Lookup.minorType == "location_key"}
):tag
-->
:tag.Location =
   {rule=Location_key2}
```

3.8 Coreferencer

The coreferencer performs orthographic co-reference between named entities in the text, e.g., *James Smith* and *Mr. Smith*. It has a set of hand-crafted rules, some of which apply for all types of entities, while others apply only for specific types, such as persons or organisations.

The coreferencer has been made Unicode-aware, so it is capable of handling multiple languages. It also handles case-sensitivity: it can be made to ignore case for all rules, with a centralised parameter, which has led to some reduction in the number of rules needed.

The rules that apply for all types of named entities in German are:

- *exact match*: two identical occurrences of the same named entity corefer.
- *equivalent*, as defined in a synonym list. This synonym list can be extended easily with new equivalent names without recompiling the orthomatcher.
- *spurious,* as defined in a list of spurious names. This rule prevents matching entities which have similar names but are otherwise different. Most frequently this is needed for companies, where a daughter company's name contains the parent company name, e.g., *BT Cellnet* and *BT*.

Some of the rules that apply to organisations and persons are:

- *word token match*: do all word tokens match, ignoring punctuation and word order, e.g., *Kalina Bontcheva* and *Bontcheva, Kalina*.
- *first token match*: does the first token in one name match the first token in the other, e.g., *Peter Smith* and *Peter*.
- *acronyms* (organisations only): handles acronyms like *International Business Machines* and *IBM*.
- *last token match*: does the last token in one name match the other name (which must be one token only), e.g., *John Smith* and *Smith*.
- *abbreviations*: matches organisation names, where one name is an abbreviation of the other, e.g., *Pan American* and *Pan Am*.
- *multi-word name matching*: there are several new rules for matching person names when they have more than two tokens and also multi-word company names. The most important rule here is whether all tokens in the shorter name match tokens in the longer name, e.g., *Second Force Recon Company* and *Force Recon Company*.

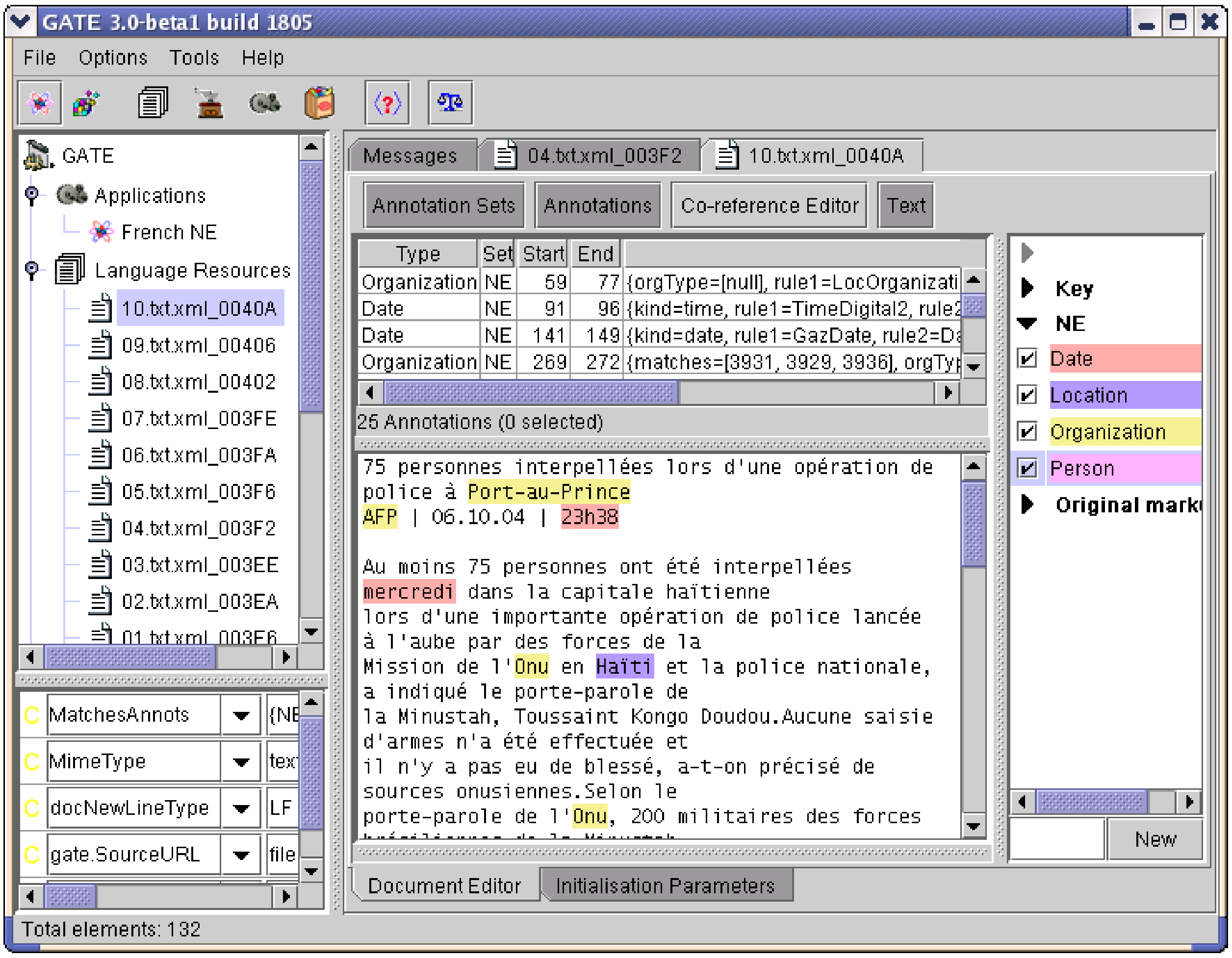
Note that the examples have been given in English for understandability, but the rules apply to German entities in exactly the same way, e.g. matching ”Herr Funkel” with ”Dieter Funkel” and so on.

The German application has been evaluated on a corpus of texts. This is detailed, along with the evaluation of the French application, in Section 7.

4 French language processing components

4.1 Architecture

The French application is similar to the German application in structure, and consists of the following processing resources:



Type	Set	Start	End	
Organization	NE	59	77	{orgType=[null], rule1=LocOrganizati
Date	NE	91	96	{kind=time, rule1=TimeDigital2, rule2
Date	NE	141	149	{kind=date, rule1=GazDate, rule2=Da
Organization	NE	269	272	{matches=[3931, 3929, 3936], orgTyp

25 Annotations (0 selected)

```
75 personnes interpellées lors d'une opération de
police à Port-au-Prince
AFP | 06.10.04 | 23h38

Au moins 75 personnes ont été interpellées
mercredi dans la capitale haïtienne
lors d'une importante opération de police lancée
à l'aube par des forces de la
Mission de l'Onu en Haïti et la police nationale,
a indiqué le porte-parole de
la Minustah, Toussaint Kongo Doudou.Aucune saisie
d'armes n'a été effectuée et
il n'y a pas eu de blessé, a-t-on précisé de
sources onusiennes.Selon le
porte-parole de l'Onu, 200 militaires des forces
```

C	MatchesAnnots	{NE
C	MimeType	tex
C	docNewLineType	LF
C	gate.SourceURL	file

Total elements: 132

Figure 2: Annotated French text in GATE

- French tokeniser
- multilingual sentence splitter
- French part-of-speech (POS) tagger
- English gazetteer
- French gazetteer
- French semantic tagger
- multilingual coreferencer

Figure 2 shows a French text in GATE annotated with various annotation types.

4.2 Tokeniser

The multilingual tokeniser used for French is exactly the same as the one used for German (see Section 3.2), except that it contains an additional post-processing resource which modifies some of the existing tokens generated by the tokeniser. The reason for this is that the POS tagger expects words to be tokenised in a certain way, and there are some differences between this and the way the GATE tokeniser works. For example, the GATE tokeniser considers words joined with an apostrophe, and the apostrophe itself, to be separate tokens, e.g. *d'* is counted as 2 tokens, whereas the tagger expects this to be considered as a single token. The post-tokeniser is a JAPE grammar which retokenises such words and maps the features appropriately from the old Tokens to the new Tokens, for example using a rule which joins a final apostrophe with the preceding word to make a single token. The methodology for this is based on that used for English (see the GATE User Guide [4] for more details).

4.3 French Sentence Splitter

The sentence splitter splits the text into sentences, as described in Section 3.3. The splitter is derived from the English splitter found in GATE, but uses a list of common French abbreviations in order to prevent the erroneous termination of a sentence after such an abbrevation, where the full stop does not denote the sentence end but is part of the abbreviation. For the sample texts, we created an alternative version of the splitter which allows a full stop not to be followed by a space, because the texts in question followed this format.

4.4 French POS tagger

The French POS tagger PR is identical to the German POS tagger PR described in Section 3.4, except that it has been trained for French. As with the German model, the French TreeTagger only works in GATE under Linux and not Windows. The wrapper for the TreeTagger generates a new annotation called TreeTaggerToken, with the features length, string and category (which contains the POS generated by the TreeTagger). The reason why the POS features are not simply added to the existing Token annotations produced by the tokeniser, as the English POS tagger in GATE does, is because the Tokens produced by the tokeniser are not in all cases the same as those expected by the TreeTagger, so this can lead to problems. It is therefore better to generate a new annotation.

Some documentation about the TreeTagger can be found at http://www.ims.uni-stuttgart.de/projekte/corplex/TreeTagger/DecisionTreeTagger.html.

The following set of POS tags is used by the tagger:

`ABR`	`abbreviation`
`ADJ`	`adjective`
`ADV`	`adverb`
`DET:ART`	`article`
`DET:POS`	`possessive pronoun (ma, ta, ...)`

INT	interjection
KON	conjunction
NAM	proper name
NOM	noun
NUM	numeral
PRO	pronoun
PRO:DEM	demonstrative pronoun
PRO:IND	indefinite pronoun
PRO:PER	personal pronoun
PRO:POS	possessive pronoun (mien, tien, ...)
PRO:REL	relative pronoun
PRP	preposition
PRP:det	preposition plus article (au,du,aux,des)
PUN	punctuation
PUN:cit	punctuation citation
SENT	sentence tag
SYM	symbol
VER:cond	verb conditional
VER:futu	verb future
VER:impe	verb imperative
VER:impf	verb imperfect
VER:infi	verb infinitive
VER:pper	verb past participle
VER:ppre	verb present participle
VER:pres	verb present
VER:simp	verb simple past
VER:subi	verb subjunctive imperfect
VER:subp	verb subjunctive present

4.5 English gazetteer

The English gazetteer is identical to the English gazetteer described earlier in Section 3.5 that is used in the German application.

4.6 French gazetteer

The French gazetteer is like the English gazetteer but contains elements specific to French. Not all English lists need translating into (or equivalent finding in) French: for example, many names of people, locations etc remain the same in both languages. However, there are also many words such as times, dates, numbers, and keywords, which must be provided in French[1]. All French lists have

[1]We are grateful to Ren Witte from the CLaC lab at Concordia University for providing us with some of the French lists

the feature ”language” with value ”French”, so that they can be identified as such if necessary.

4.7 French semantic tagger

The French semantic tagger is similar in design to the German semantic tagger described in Section 3.7, except that it does not contain the compound analysis component, because French is not a compounding language. The tagger was, like the German tagger, adapted from the English semantic tagger available with GATE. It consists of a set of 20 phases run sequentially. There is one phase for each annotation type found, and several intermediate phases that identify keywords and other temporary annotations which may be modified later in the presence of more contextual information.

4.8 Coreferencer

The coreferencer used for French is exactly the same as that used for German, described in Section 3.8, as the rules work as well for French as they do for English and German.

5 Spanish language processing components

5.1 Architecture

The Spanish application currently consists of the following processing resources:

- multilingual tokeniser
- multilingual sentence splitter
- Spanish POS tagger
- Spanish keyword annotator

Other components for Spanish are currently being developed (see Section 9 for more details).

5.2 Tokeniser and Splitter

The tokeniser and sentence splitter used are exactly the same as those used for German and French described in Sections 3.2 and 3.3. The Spanish splitter should ideally have a set of abbreviations specific to Spanish, but this has not yet been implemented.

5.3 POS tagger

We have incorporated a Spanish POS tagger based on the Brill tagger, but trained on Spanish text. The tags for Spanish are different from the ones for English, and are defined in [2]. We do not currently have any indication of the quality of the results, however.

5.4 Keyword annotator

The keyword annotator is a GATE processing resource currently under development at ISOCO. Its aim is to find technical terms that can be found in the legal case study to be performed in SEKT[2]. It uses a statistical algorithm based on [11] to detect the most relevant words in a text. The algorithm calculates the distances between the occurrences of each word and assigns a higher score to those that deviate more from the normal distribution. Words with the highest score are marked with the ”Keyword” annotation, and their score is recorded as an attribute of the annotation. The PR requires a tokeniser and sentence splitter to have been run previously on the text.

In its current state, the component shows some inaccurate behaviour, that will improve as more components that can provide linguistic information are developed, such as a Spanish stopwords detector, Spanish lemmatiser, or Spanish synonyms.

6 Stemmers

We have also created a set of stemmers as PRs in GATE for the following 11 European languages: Danish, Dutch, English, Finnish, French, German, Italian, Norwegian, Portuguese, Russian, Spanish and Swedish. These take the form of wrappers for the Snowball stemmers freely available from http://snowball.tartarus.org. Each Token is annotated with a new feature stem, with the stem for that word as its value.

6.1 Algorithms

The stemmers are based on the Porter stemmer for English [13], with rules implemented in Snowball e.g.

```
define Step_1a as
( [substring] among (
 ’sses’ (<-’ss’)
’ies’ (<-’i’)
’ss’ () ’s’  (delete)
 )
```

[2]Note that there are no keyword annotators for French and German because this component is only required for the legal case study, which is not being performed on these languages.

The stemmers have not been evaluated, however, Tomlinson [15] compared the original Snowball (algorithmic) stemmers with a commercial lexical stemming (lemmatization) system. Of the nine languages tested, six gave differences that were not statistically signifant, two did better under the lemmatization system (Finnish and German), and one better under Snowball (Swedish).

The stemmers have not been integrated in applications because it is unclear at the moment how they would contribute. In other words, we could make use of stemmed words in the semantic tagger component, but this requires more extensive work which we have not yet done. Furthermore, the stemmers actually do overstemming, in that they create stems that are not actual words. This approach works for Information Retrieval purposes, but it can be too limiting for more linguistically oriented purposes such as we have in mind for the applications developed here. We include the stemmers, however, because they can be used on their own or combined with other components for other purposes such as aiding Information Retrieval.

7 Evaluation

We evaluated the results of the applications using the standard Information Retrieval (IR) measures of Precision, Recall and F-measure [16]. These measures have been used in many Information Extraction (IE) evaluations such as MUC (Message Understanding Conferences) [1]. The work on evaluation has been carried out in conjunction with work as part of the EU-funded Network of Excellence KnowledgeWeb (IST-2004-507482). Discussion of evaluation metrics for information extraction and semantic annotation for research and industry can be found in the KnowledgeWeb deliverables D2.1.4 (Benchmarking Ontology Tools) and D1.2.3 (Methods for ontology evaluation), and are also planned for future SEKT deliverables.

Precision and Recall are calculated by first measuring the number of Correct, Partially Correct, Spurious and Missing annotations. An annotation is considered partially correct if the annotation type is correct but the extent is wrong, for example if ”Smith” instead of ”Mr Smith” were matched as a Person.

Precision measures the number of correctly identified items as a percentage of the number of items identified. In other words, it measures how many of the items that the system identified were actually correct, regardless of whether it also failed to retrieve correct items. The higher the precision, the better the system is at ensuring that what is identified is correct. Precision is defined formally as:

$$Precision = \frac{Correct + 1/2 Partial}{Correct + Spurious + Partial} \tag{1}$$

Recall measures the number of correctly identified items as a percentage of the total number of correct items. In other words, it measures how many of the items that should have been identified actually were identified, regardless of how many spurious identifications were made. The higher the recall rate, the better the system is at not missing correct items. Recall is defined formally as:

$$Recall = \frac{Correct + 1/2Partial}{Correct + Missing + Partial} \tag{2}$$

Clearly, there must be a tradeoff between precision and recall, for a system can easily be made to achieve 100% precision by identifying nothing (and so making no mistakes in what it identifies), or 100% recall by identifying everything (and so not missing anything). The **F-measure** [16] is often used in conjunction with Precision and Recall, as a weighted average of the two. F-measure is defined formally as:

$$F - measure = \frac{(\beta^2 + 1)P * R}{(\beta^2 R) + P} \tag{3}$$

where β reflects the weighting of P vs. R.

A simpler version of this is to use the equation:

$$F - measure = \frac{P * R}{0.5 * (P + R)} \tag{4}$$

where P and R are given equal weights.

7.1 Benchmarking tool

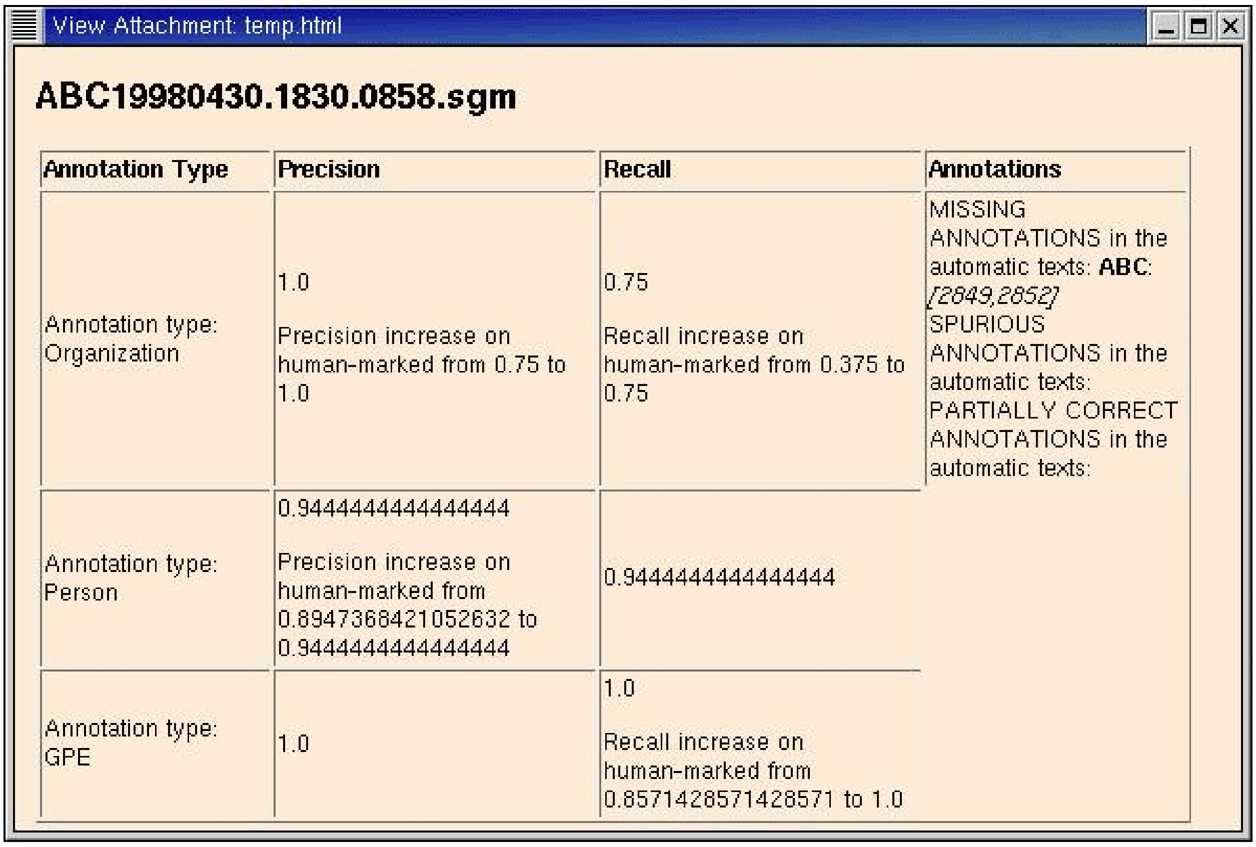
The evaluation was carried out with a tool in GATE specifically designed for this kind of evaluation, known as the corpus benchmark tool. This tool enables evaluation to be carried out over a whole corpus and also enables tracking of the system's performance over time.

The tool requires a clean version of a corpus (with no annotations) and an annotated reference corpus. The tool can be run in generation mode to produce a set of texts annotated by the system (which we call the "stored set"). As the name indicates, these texts are stored for future use. There should also be a reference set, consisting of gold standard annotations (created manually). If the application is modified after creating the stored set, we can use the ”latest set”, which is a more recent version of the results than the stored set, and we can compare the two to see which is better.

The tool can be run in three ways:

1. comparing the stored set with the reference set;
2. comparing the stored set with the latest set;
3. comparing the latest set with the reference set.

In each case, performance statistics will be output for each text in the set, and overall statistics for the entire set, in comparison with the reference set. In case 2, information is also provided about whether the figures have increased or decreased in comparison with the stored set. The stored set can be updated at any time by rerunning the tool in generation mode with the latest version of the system resources. Furthermore, the system can be run in verbose mode, where for each Precision and Recall figure below a certain threshold (set by the user), the non-coextensive annotations (and their corresponding text) will be displayed. The output of the tool is written to an HTML file in tabular form, for easy viewing of the results, as depicted in Figure 3.


View Attachment: temp.html

ABC19980430.1830.0858.sgm

Annotation Type	**Precision**	**Recall**	**Annotations**
Annotation type: Organization	1.0 Precision increase on human-marked from 0.75 to 1.0	0.75 Recall increase on human-marked from 0.375 to 0.75	MISSING ANNOTATIONS in the automatic texts: **ABC**: *[2849,2852]* SPURIOUS ANNOTATIONS in the automatic texts: PARTIALLY CORRECT ANNOTATIONS in the automatic texts:
Annotation type: Person	0.9444444444444444 Precision increase on human-marked from 0.8947368421052632 to 0.9444444444444444	0.9444444444444444	
Annotation type: GPE	1.0	1.0 Recall increase on human-marked from 0.8571428571428571 to 1.0	

Figure 3: Fragment of results from benchmark tool

7.2 Evaluation of German application

The evaluation corpus consisted of 20 German news texts from the CONLL collection (http://cnts.uia.ac.be/conll2002/ner/). The entities considered are the same as those recognised by the system, i.e. Person, Organization, Location, Date, Time, Money and Percent. Table 1 shows the results. Considering the early stage of development of the application, these results are very promising.

From the results so far, we can see that the recall is a bit low in some cases, although the precision is generally very good (except in the case of Organization). This is largely because more work is needed on the gazetteer lists and on the compound analysis module (particularly for Organizations which are often formed by compounding).

Type	Correct	Partial	Missing	Spurious	P	R	F
Person	8	3	16	2	73.1	35.2	47.5
Organization	4	2	47	5	45.5	9.4	15.6
Location	47	2	11	22	67.6	80	73
Date/Time	29	6	1	72.7	88.9	80	
Percent	3	0	0	0	100	100	100
Total	91	13	75	102	75.9	65.6	60.9

Table 1: Results of evaluation of German application

Type	Correct	Partial	Missing	Spurious	P	R	F
Person	25	9	8	5	75.6	70.2	72.8
Organization	25	5	28	14	62.5	47.4	53.9
Location	30	8	22	0	89.5	56.7	69.4
Date/Time	28	7	7	13	65.6	75	70
Total	108	29	65	32	72.9	64.4	68.4

Table 2: Results of evaluation of French application

7.3 Evaluation of French application

The evaluation corpus consisted of 20 French news texts from the online newspaper ”Le Monde”, downloaded from the web and annotated by hand using the GATE annotation tool. The entities considered are the same as those recognised by the system, i.e. Person, Organization, Location, Date/Time, Money and Percent. Table 2 shows the results for Precision, Recall and F-measure. Considering the early stage of development of the application, these results are very promising. On the test set we used, there were no key annotations for Money or Percent, so we ignored these in the evaluation.

From the results table we can see here that performance is generally very good, though again, the recognition of Organizations is a bit weak. This is mainly due to the lack of gazetteer lists for common French companies, organisations and locations, and will be improved in future versions.

8 Related work

Developing Information Extraction systems and language processing resources for languages other than English is by no means a new task. Knowledge engineering approaches such as that used by GATE have been used to develop existing IE systems from scratch in other languages, but it can be a time-consuming task in this case. Machine learning-based approaches using e.g. Hidden Markov Models (HMMs) make the task easier. For example, Saito and Nagata [14] use an HMM-based system to perform IE in English, Chinese, Korean and Japanese, by means of a common analytical engine which can handle any language simply by changing the lexical analysis rules and the statistical language model. In theory it can be adapted to any new language if an appropriate training corpus exists, but this is the main problem with this and other learning-based systems, namely that development or acquisition of such a corpus is not easy.

More recently, bootstrapping techniques have been used to create large amounts of noisy training data from a small set of seeds or more accurate training data, e.g. [10], who claim that a large but noisy training corpus is sufficient to train a statistical semantic tagger such as *phrag* [12]. While this approach is used to adapt an IE system to a new domain (biomedicine) rather than to a new language, the same principle applies.

Another method of adapting IE systems to new domains is to use an idea called *annotation projection*, which involves using a parallel corpus to train a system on a new language. The basic idea behind this is to create a word-aligned parallel corpus of the two languages, tag the data on the source language (e.g. with semantic annotations) using the source language application, then project the annotations onto the equivalent words in the target language. Some post-processing is usually necessary at this point, but the target language corpus can then serve as a source of annotated training data for the target language application. Of course, the main problem with this method lies in having a suitable parallel corpus and alignment tools. Also, it relies on the source language annotations being very accurate, otherwise errors can be propagated to the new language. So generally some manual post-editing of the source language is necessary, which can be time-consuming if a large corpus is used. This technique has largely been used for POS-tagging and parsing, e.g. [17, 6], but there is currently ongoing research in applying this to semantic annotation.

9 Future work

The work presented in this deliverable is ongoing, and further improvements and extensions can be expected in the next phase of the project. As discussed earlier, one of the problems with the TreeTagger is that it is only freely available for use under Linux. We therefore plan to either make a version of the French and German applications which does not use a tagger, or to incorporate a different tagger as an alternative. If we make a version without a tagger, the grammar rules must be adjusted to make use of other information such as orthography (e.g. by assuming that words beginning with a capital letter are proper nouns in the case of French, or any type of nouns in the case of German). We can, however, expect some loss of accuracy with such an approach.

Other planned work involves improving the gazetteer lists and grammar rules in the French and German applications, adding a Spanish POS tagger and possibly a Spanish semantic tagger, and making better use of the coreferencer in order to match unknown words with known entities (Persons, Locations, etc.).

References

[1] Nancy Chinchor. Muc-4 evaluation metrics. In *Proceedings of the Fourth Message Understanding Conference*, pages 22–29, 1992.

[2] M. Civit. Gua para la anotacin morfosintctica del corpus CLiC-TALP. Technical report, X-TRACT Working Paper, Universitat de Barcelona, 2000.

[3] H. Cunningham, D. Maynard, K. Bontcheva, and V. Tablan. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. In *Proceedings of the 40th Anniversary Meeting of the Association for Computational Linguistics (ACL'02)*, 2002.

[4] H. Cunningham, D. Maynard, K. Bontcheva, V. Tablan, C. Ursu, M. Dimitrov, M. Dowman, N. Aswani, and I. Roberts. *Developing Language Processing Components with GATE Version 3 (a User Guide)*. `http://gate.ac.uk/`, 2005.

[5] H. Cunningham, D. Maynard, and V. Tablan. JAPE: a Java Annotation Patterns Engine (Second Edition). Research Memorandum CS–00–10, Department of Computer Science, University of Sheffield, November 2000.

[6] R. Hwa, P. Resnik, A. Weinberg, and O.. Kolak. Evaluating translational correspondence using annotation projection. Technical report, University of Maryland, 2002.

[7] D. Maynard, K. Bontcheva, and H. Cunningham. Automatic Language-Independent Induction of Gazetteer Lists. In *Proceedings of 4th Language Resources and Evaluation Conference (LREC'04)*, 2004.

[8] D. Maynard and H. Cunningham. Multilingual Adaptations of a Reusable Information Extraction Tool. In *Proceedings of the Demo Sessions of EACL'03*, Budapest, Hungary, 2003.

[9] D. Maynard, V. Tablan, K. Bontcheva, and H. Cunningham. Rapid customisation of an Information Extraction system for surprise languages. *Special issue of ACM Transactions on Asian Language Information Processing: Rapid Development of Language Capabilities: The Surprise Languages*, 2003.

[10] A. Morgan, L. Hirschman, A. Yeh, and M. Colosimo. Gene Name Extraction Using FlyBase Resources. In *Proc. of ACL 2003 Workshop on Natural Language Processing in Biomedicine*, pages 1–8, Sapporo, Japan, 2003.

[11] M. Ortuo, P. Carpena, P. Bernaola-Galvn, E. Muoz, and A. M. Somoza. Keyword detection in natural languages and dna. *Europhysics Letters*, 57(5):759–764, 2002.

[12] D. Palmer, J. Burger, and M. Ostendorf. Information extraction from broadcast news speech data. In *Proceedings of the DARPA Broadcast News and Understanding Workshop*, 1999.

[13] M.F. Porter. An algorithm for suffix stripping. *Program*, 14(3):130–137, 1980.

[14] K. Saito and M. Nagata. Multi-Language Entity Recognition System based on HMM. In *ACL Workshop on Multilingual and Mixed-language Named Entity Recognition: Combining Statistical and Symbolic Models*, Sapporo, Japan, 2003.

[15] Stephen Tomlinson. Lexical and algorithmic stemming compared for 9 European languages with Hummingbird SearchServer(TM) at CLEF 2003. In Carol Peters, editor, *Working notes for the CLEF 2003 Workshop*, pages 22–29, Trondheim, Norway, 2003.

[16] C.J. van Rijsbergen. *Information Retrieval*. Butterworths, London, 1979.

[17] D. Yarowsky and G. Ngai. Inducing multilingual POS taggers and NP bracketers via robust projection across aligned corpora. In *Proceedings of NAACL-01*, 2001.