Corpora and evaluation tools for multilingual named entity grammar development

Christian Bering[1], Witold Drozdzynski[2], Gregor Erbach[1], Clara Guasch[3], Petr Homola[1], Sabine Lehmann[3], Hong Li[2], Hans-Ulrich Krieger[2], Jakub Piskorski[2], Ulrich Schäfer[2], Atsuko Shimada[2], Melanie Siegel[1], Feiyu Xu[2], Dorothee Ziegler-Eisele[1]

[1] Saarland University, Computational Linguistics Department, Saarbrücken
[2] DFKI GmbH, Language Technology Lab, Saarbrücken
[3] Acrolinx GmbH, Berlin

Abstract

We present an effort for the development of multilingual named entity grammars in a unification-based finite-state formalism (SProUT). Following an extended version of the MUC7 standard, we have developed Named Entity Recognition grammars for German, Chinese, Japanese, French, Spanish, English, and Czech. The grammars recognize person names, organizations, geographical locations, currency, time and date expressions. Subgrammars and gazetteers are shared as much as possible for the grammars of the different languages. Multilingual corpora from the business domain are used for grammar development and evaluation. The annotation format (named entity and other linguistic information) is described. We present an evaluation tool which provides detailed statistics and diagnostics, allows for partial matching of annotations, and supports user-defined mappings between different annotation and grammar output formats.

1 Introduction

Motivation

Named Entity Recognition (NER) is a fundamental technology for a number of advanced information management applications, including search engines, question answering systems, customer relationship management (CRM) systems, text mining and business intelligence. All of these applications can benefit from accurate NER. For example, search engines and question answering systems can be enhanced by allowing searches and questions for particular persons, companies or locations. In CRM applications, such as automatic e-mail answering, accurate NER will enable a link between customer e-mails and data stored in a customer database. In text mining, accurate NER will allow the construction of databases with information extracted about particular entities.

Multilingual NER is important for globalised CRM, e.g., automatic e-mail categorization and answering for customers from different linguistic communities. Other applications of multilingual NER are cross-language information retrieval, and business intelligence applications, where information about a particular person or company has to be extracted from textual sources in different languages.

Challenges

There are a number of challenges in multilingual information extraction. With non-Western-European languages, the issue of different alphabets and character sets has to be dealt with. Different languages have different tokenization conventions. For example, some languages (German) use a blank space between the dash (Gedankenstrich) and the surrounded words, while others attach the dash without a space (Spanish). However, in German a dash attached to a word normally indicates a split compound ("Ein- und Ausgang" for "Eingang und Ausgang", entry and exit). Other issues are different number (decimal point vs. comma), time, and currency formats.

There is a large overlap between different languages within organization names. Some names and acronyms such as "New York", "George Bush", or "IBM" are identical in most languages, although they may carry different inflections. They enable the re-use of lexicons and gazetteers across different languages. In other cases there are differences, e.g., "London" vs. "Londres", "Firenze" vs. "Florence" vs. "Florenz", "NATO" vs. "OTAN", or Pope "Johannes Paul" vs. "John Paul". It would be tempting to include all variants into a single list, but in some cases this leads to undesired ambiguities, e.g. "München" (German) vs. "Munich" (English) vs. "Monaco" (Italian). Including the last form in the German grammar would associate the wrong reference to the string "Monaco".

We try to re-use linguistic resources as much as possible while avoiding the mentioned problems.

Our Approach

In order to address the challenges of multilingual NER, we have employed a language-independent finite-state formalism [Becker et. al, 2002], in which language resources and processing components (such as morphological analyzers) for different languages are integrated (see section 2). All languages

use the same tokenizer and token classes. In order to ensure consistency and maintainability of multilingual grammars, we make use of shared resources (lexicon, gazetteer, and grammar rules). To facilitate the usage of NER results across languages, the grammars for all languages produce a uniform language-independent output structure, for which type definitions have been written in TDL.

Multilingual Named Entity Corpora

Corpora with annotated named entities are essential for developing and evaluating NE grammars. We try to re-use existing corpora (e.g., from the Message Understanding Conference MUC (Chinchor, 1997), plus other sources) as much as possible, but also annotate corpora for languages for which no suitable NE-annotated corpora are available.

The annotation of the corpora may differ from the output structures produced by the grammar in several crucial aspects:

- use of different classes of named entities, or different granularities (e.g., *organization* and its subclasses *company, university, government etc.*)
- the extent of a NE may be different: e.g., a person name may or may not include a function and a title ("President George Bush" vs. "George Bush")
- the markup of the corpus may be textually oriented (e.g., as XML tags) while output of grammar is a semantic structure

Such differences may arise because

- existing corpora which were developed with a different purpose are re-used,
- the output structure of the grammar is changed after corpora were annotated,
- the corpora are annotated for multiple uses, of which NER is only one.

These differences pose challenges for testing and evaluation of grammars with respect to a corpus, since a NE may be recognized correctly according to the intentions of the grammar developer, but may be annotated differently in the corpus. In order to address such mismatches, we have developed a diagnostic and evaluation tool, which allows for user-defined mappings between different NE classes, for controlled partial overlap between recognized and annotated NEs, and supports user-defined mappings between text-based and semantically-based annotations and output structures.

2 SProUT: A Unification-Based Finite-State Toolkit

In this section we describe SProUT, a platform for the development of multilingual text processing components, which constitutes the underlying framework for NE grammar development. Considering construction of efficient and domain-adaptive text-processing systems, the main motivation for developing *SProUT* comes from the need to have a system that (i) allows a flexible integration of different processing modules and (ii) to find a good trade-off between processing efficiency and expressiveness. On the one hand, very efficient *finite state* devices have been successfully applied to real-world applications. On the other hand, *unification-based grammars* (UBGs) are designed to capture fine-grained syntactic and semantic constraints, resulting in better descriptions of natural language phenomena. In contrast to finite state devices, unification-based grammars are also assumed to be more transparent and more easily modifiable. Our idea now is to take the best of these two worlds, having a finite state machine that operates on typed feature structures (TFSs). I.e., transduction rules in *SProUT* do not rely on simple atomic symbols, but instead on TFSs, where the left-hand side (LHS) of a rule is a regular expression over TFSs, representing the recognition pattern, and the right-hand side (RHS) is a sequence of TFSs, specifying the output structure. Consequently, equality of atomic symbols is replaced by *unifiability* of TFSs and the output is constructed using TFS *unification* w.r.t. a type hierarchy.

2.1 XTDL – The Formalism of *SProUT*

XTDL combines two well-known frameworks: typed feature structures and regular expressions. XTDL is defined on top of TDL, a definition language for TFSs (Krieger and Schäfer, 1994) that is used as a descriptive device in several grammar systems (LKB, PAGE, PET). We use the following fragment of TDL, including coreferences and functional application.

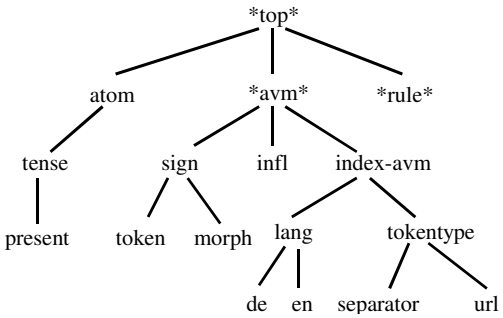
```
type-def ::= type { avm-def | sub-def } "."
type     ::= identifier
sub-def  ::= ":<" type
avm-def  ::= ":=" avm
avm      ::= term { "&" term }*
term     ::= type | fterm | string | coref
fterm    ::= "[" [ attr-val { "," attr-val }* ] "]"
attr-val ::= attribute avm
```

```
attribute ::= identifier
coref     ::= "#"identifier
```

Apart from the integration into the rule definitions below, we also employ this fragment in SProUT for the establishment of a type hierarchy of linguistic entities. In the example definition below, the morph type inherits from sign and introduces three more morphologically motivated attributes with the corresponding typed values.

```
morph := sign & [ POS  atom, STEM atom, INFL infl ].
```

The next figure depicts a fragment of the type hierarchy used in the example.



A rule in XTDL is defined as a recognition pattern for the LHS, written as a regular expression, and an output description on the RHS. A named label serves as a handle to the rule. Regular expressions over TFSs describe sequential successions of linguistic signs. We provide a couple of standard operators; see the EBNF below. Concatenation is expressed by consecutive items. Disjunction, Kleene star, Kleene plus, and optionality are represented by the operators |, *, +, and ?, resp. {n} after an expression denotes an n-fold repetition. {m,n} repeats at least m times and at most n times.

```
rule    ::= identifier ":>" regexp "->" {fterm}* [fun-op]"."
regexp::= avm | "(" regexp ")" | regexp {regexp}+ | regexp "|" {regexp}+ | regexp { "*" | "+" | "?" } |
          regexp "{" int [ "," int ] "}"
fun-op    ::= "where" { coref "=" fun-app }+
fun-app   ::= identifier "(" term { "," term }  ")"
```

The XTDL grammar rule below illustrates the syntax. It describes a sequence of morphologically analyzed tokens (of type morph). The first TFS matches one or zero items (?) with part-of-speech Determiner. Then, zero or more Adjective items are matched (*). Finally, one or two Noun items ({1,2}) are consumed. The use of a variable (e.g., #1) in different places establishes a coreference between features. This example enforces, e.g., agreement in case, number, and gender for the matched items. I.e., all adjectives must have compatible values for these features. Eventually, the description on the RHS creates a feature structure of type phrase, where the category is coreferent with the category Noun of the right-most token(s) and the agreement features result from the unification of the agreement features of the morph tokens

```
np :>
  (morph & [  POS  Determiner, INFL  [CASE #1, NUM #2, GEN #3 ]] )?
  (morph & [  POS  Adjective, INFL  [CASE #1, NUM #2, GEN #3 ]] )*
  (morph & [  POS  Noun & #4, INFL  [CASE #1, NUM #2, GEN #3 ]] ){1,2}
 -> phrase & [CAT #4, AGR agr & [CASE #1, NUM #2, GEN #3 ]].
```

The choice of TDL has a couple of advantages. TFSs as such provide a rich descriptive language over linguistic structures and allow for a fine-grained inspection of input items. They represent a generalization over pure atomic symbols. Unifiability as a test criterion, whether a transition is viable, can be seen as a generalization over symbol equality. Coreferences in feature structures express structural identity. Their properties are exploited in two ways. They provide a stronger expressiveness since they create dynamic value assignments on the automaton transitions and thus exceed the strict locality of constraints in an atomic symbol approach. Furthermore, coreferences serve as a means of information transport into the output description on the RHS of the rule. Finally, the choice of feature structures as primary citizens of the information domain makes composition of modules very simple, since input and output are all of the same abstract data type.

2.2 System Description

The core of the SProUT system is comprised of following components:

- a finite-state machine toolkit for building, combining and optimizing various types of finite-state devices,
- a flexible XML-based regular compiler for converting regular patterns into their corresponding compressed finite-state representation (Piskorski et al., 2002),
- the JTFS package which provides standard operations for constructing and manipulating TFSs (e.g., unification), and
- an XTDL grammar interpreter.

The grammar interpretation is divided into two steps. Firstly, regular patterns (LHS of rules) are employed to match text fragments using solely unifiability to filter the potential candidates for the space consuming unification. Secondly, appropriately instantiated LHS patterns are used for the construction of the output structures via unification (RHS). Since the output of the interpreter are again TFSs, the result can be used to feed (higher-level) linguistic processing components. In this way, SProUT supports cascaded architectures straightforwardly.

Currently, the system provides three online linguistic processing components, including a tokenizer, gazetteer, and a morphological analyzer. The tokenizer maps character sequences to tokens and performs fine-grained token classification. The task of gazetteer is recognition of named entities based on static named-entity lexica. The morphology unit provides lexical resources for English, German[1], French, Italian, and Spanish which were compiled from the full-form lexica of MMorph[2] (Petitpierre and Russell, 1995). For Asian language, we integrated Chasen (Asahara and Matsumoto, 2000) for Japanese and Shanxi (Liu, 2000) for Chinese. The backbone architecture of SProUT is depicted in figure 1.

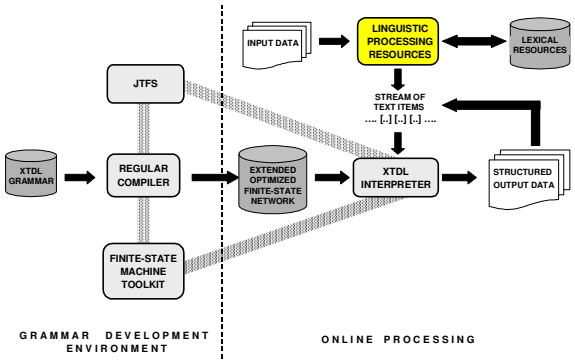


Figure 1. Architecture of SProUT

SProUT comes with a user-friendly grammar development environment. The development process is started with a project definition. A project is defined as a specific application in SProUT, e.g., NER or template filling (IE). A project consists of a grammar definition and a system configuration. Given a new project (see figure 2), an offline defined type hierarchy based on TDL has to be loaded. A grammar developer can navigate through the type hierarchy. An XTDL grammar can consist of more than one grammar file (see left upper frame in figure 2). The grammar rules are listed in the left down frame. Rule names can be assigned colors for targeted visualization of searching results.

The grammarians can switch between three views of the edited grammars: XML, XTDL and equation. The equation mode presents graphical visualization of the rules; see the right upper frame in figure 2. SProUT automatically converts one format to another.

System configuration allows the specification of the individual processing components and corresponding resources required by the grammar. The grammar can be tested against an input text, where the instantiated rules and matched text spans are highlighted. An example of testing named entity grammars for Japanese is demonstrated in figure 3. The current system is implemented in Java and C++, and runs on both MS Windows and Linux OS.

[1] The German morphology is equipped with an online shallow compound recognition.
[2] An additional compaction of the original MMorph was performed by substituting special readings through more general ones using type generalization and subsumption checking.

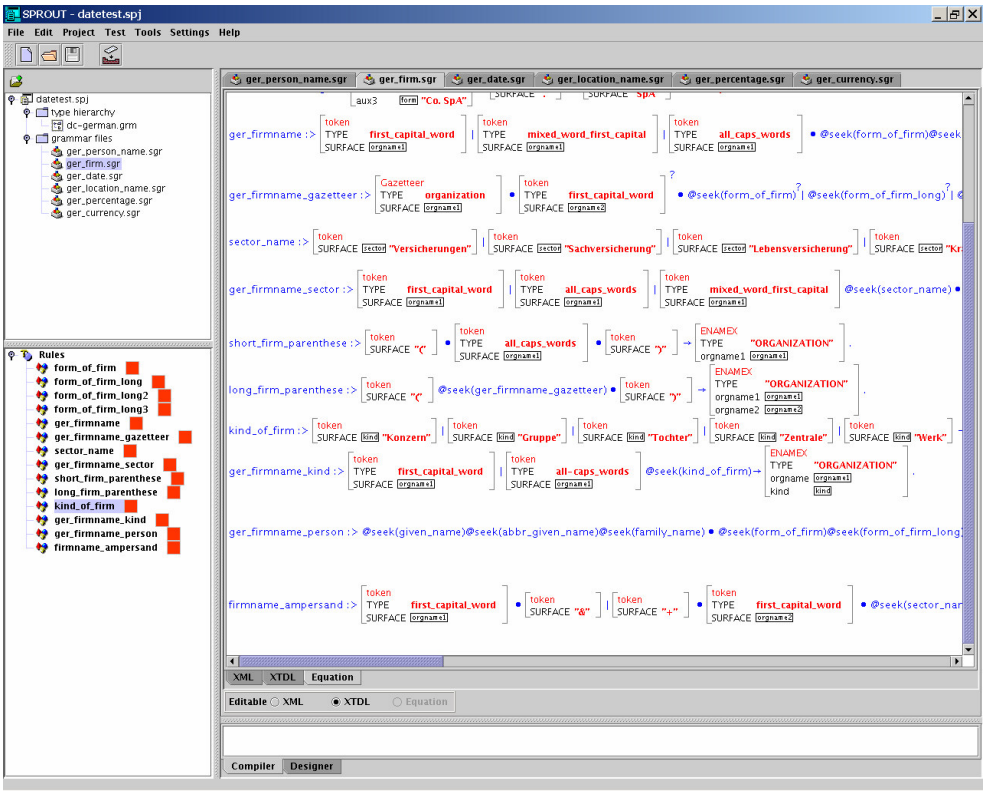


Figure 2. The grammar development environment of SProUT

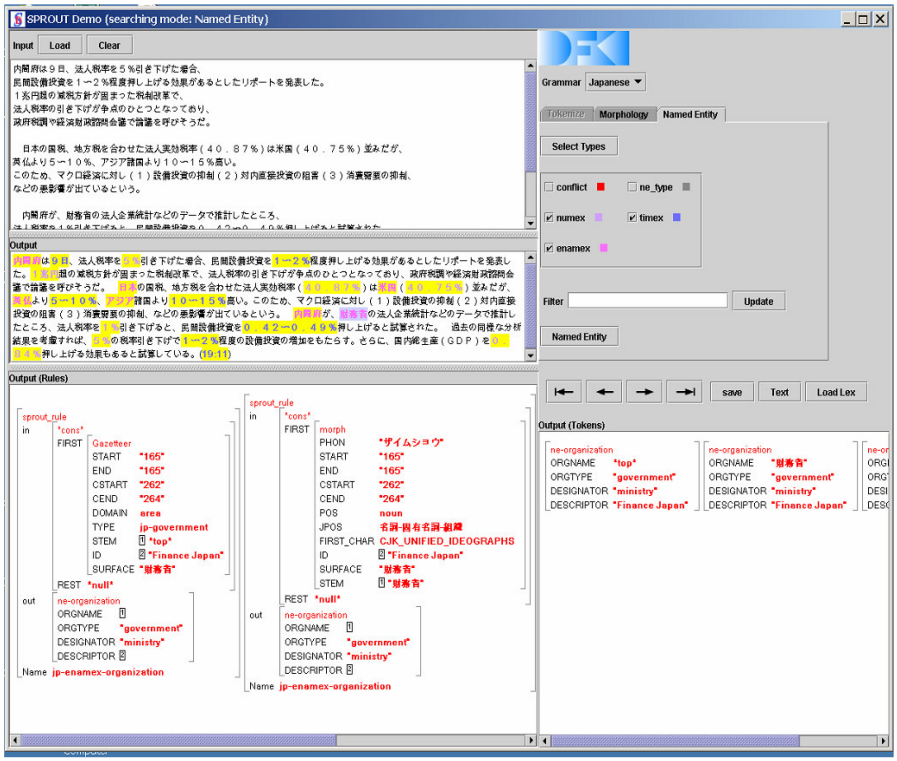


Figure 3. The multilingual NER application

3 Multilingual Grammar Development

A guiding principle for our multilingual named entity grammar development is maximal sharing of resources across different languages. Token classes, output structures and grammar fragments are shared for different languages, improving the maintainability and consistency of linguistic resources.

Shared Token classes

NER makes heavy use of surface-oriented features for recognizing particular kinds of NEs. For example, a string consisting entirely of upper-case letters is likely to be an acronym (used in grammars for organization names), while a string consisting of four digits is likely to be a year (used in grammars for date expressions). We make use of special token classes and subclasses for these kinds of strings.

Different languages have different tokenization conventions. For example, some languages (German) use a blank space between the dash (Gedankenstrich) and the surrounded words, while others attach the dash without a space (Spanish). However, in German a dash attached to a word normally indicates a split compound ("Ein- und Ausgang" for "Eingang und Ausgang", entry and exit).

Despite these differences, we have agreed on a single set of token classes for the Western European languages, in order to simplify the overall system and to facilitate the sharing of grammar fragments – which make use of the token classes – across languages. The full list of token classes is given in the following. [examples for each class to be provided in the final version of the paper.]

```
any_natural_number := tokentype & [INDEX '1].
twoDigitNumber :=  any_natural_number & [SUBINDEX '2].
fourDigitNumber :=  any_natural_number & [SUBINDEX '3].
point_or_comma := tokentype &  [INDEX '2].
number_percent_composition := tokentype &  [INDEX '3].
number_dot_compositum := tokentype &  [INDEX '4].
number_word_composition := tokentype &  [INDEX '5].
digit_salsh_compound := tokentype &  [INDEX '7].
digit_dash_compositum  := tokentype &  [INDEX '8].
all_caps_words := tokentype &  [INDEX '9].
digit_colon_compositum := tokentype &  [INDEX '10].
lowercase_word := tokentype &  [INDEX '11].
first_capital_word := tokentype &  [INDEX '12].
words_slash_compound_first_capital := tokentype &  [INDEX '13].
mixed_word_first_lower := tokentype &  [INDEX '14].
words_slash_compound_first_capital := tokentype &  [INDEX '13].
mixed_word_first_lower := tokentype &  [INDEX '14].
decimal_number_with_period  := tokentype &  [INDEX '23].
decimal_number_with_coma := tokentype &  [INDEX '24].
e_mail_adress  := tokentype &  [INDEX '32].
url_adress := tokentype &  [INDEX '33].
mixed_word_first_capital := tokentype &  [INDEX '34].
separator_symbol :=  tokentype &  [INDEX '40].
digit_bsalsh_compositum := tokentype &  [INDEX '102].
complex_compound_first_capital  := tokentype &  [INDEX '103].
word_slash_compound_dash_first_capital := tokentype &  [INDEX '105].
word_slash_compound_dash_first_lower := tokentype &  [INDEX '106].
contains_digits_and_letters_and_other_symbols := tokentype &  [INDEX '122].
word_apostrophe := tokentype & [INDEX '125].
```

Shared Output Structures

The grammars for all six languages produce the same, semantically oriented output structures. The possible output structures have been defined by type definitions in TDL. A sample type definition for persons and locations is given in the following. The type definition defines the features and subtypes for each type.

```
ne_type := sign & [DESCRIPTOR string].

enamex := ne_type.

ne-person := enamex &
        [TITLE list-of-strings,
         GIVEN_NAME list-of-strings,
         SURNAME list-of-strings,
         P-POSITION list-of-strings,
         NAME-SUFFIX string].

ne-location := enamex &
        [LOCTYPE loc-type,
```

```
          LOCNAME string].

loc-type :< atom.
river := loc-type.
continent := loc-type.
country := loc-type.
province := loc-type.
city := loc-type.
```

The use of shared output structures facilitates the construction of multilingual applications, as the interface between the application and the grammar will be the same for the different languages.

Shared Grammars

One of the main advantages of the SProUT system is its emphasis on reusable and easily extensible grammars. This section shows how the possibilities offered by the linguistic engine have been used to develop multilingual parallel grammars for English, French, and Spanish.

SProUT allows us to define any number of cascaded rules and to distribute them in a number of files. This feature has been used to implement grammars for English, French and Spanish in such a way that some generic files are shared by the three languages, while others are language-specific. The basic idea here was to make use of the fact that some key element structures are identical in the three languages, e.g. date formats such as "20.10.2001" (to refer to the 20th of October 2001) are used in English, French and Spanish. This structure has been defined in a generic data grammar file which is shared by the three languages. Structures such as "20th of October 2001", "20 octobre 2001" or "20 de octubre del 2001", on the other hand, are defined in the corresponding language-specific files. This design has various advantages. In the first place, it makes the grammars easily reusable and extensible. Secondly, the grammar development turns out to be much more efficient and less time-consuming and error-prone, since in some cases the three languages can be corrected at once by simply modifying the generic structures. This modular approach to multingual grammar writing also facilitates the addition of new languages, as the experience of writing the English grammars clearly proved. After having written the generic components for French and Spanish, and the language specific grammars for both languages, the English grammars could be developed very quickly and efficiently. We believe that this method could be easily extended to other West-European languages. The following (simplified) examples illustrate how language specific grammars make use of general rules. The first rule (general_monetary_amount) specifiies the types of numeric tokens that can refer to a monetary amount. The second rule (general_currency) defines signs or acronyms that are internationally used to refer to currencies. The signs are specified as tokens, and the acronyms are listed in the gazetteer (see end of this section). Finally, general_money will match things like "$300" or "5.000 CAD", that is, a monetary amount followed by a currency sign or acronym.

```
general_monetary_amount :>
       token & [ TYPE any_natural_number ,         SURFACE #amount ]
       | token & [ TYPE decimal_number_with_coma , SURFACE  #amount ]
       | token & [ TYPE decimal_number_with_period , SURFACE #amount ]
       -> dummy.

general_currency : >  Gazetteer & [TYPE generalCurrencyAcronyms, SURFACE #currency]
               | token & [ SURFACE "¥" ,   ID #currency & "JPY" ]
               | token & [ SURFACE "$" ,      ID #currency & "USD" ]
               | token & [ SURFACE "€",      ID #currency & "EUR" ]
               | token & [ SURFACE "£",      ID #currency & "GBP"
       -> dummy.

general_money    :>
             ( @seek(general_monetary_amount)
               @seek(general_currency) )
       |
       (  @seek(general_currency)
             @seek(general_monetary_amount)  )
    ->
             money & [ CURRENCY  #currency ,
                       AMOUNT    #amount ,
                       MAGNITUDE #magnitude ].
```

In order to make sure that language specific monetary expressions were recognized, we only needed to expand the general rules with language specific contexts. Thus, en_monetary_amount below ensures that not only "300" but also the English expression "300 million" is recognized as a monetary amount.,

and en_currency adds English currency expressions (listed in the gazetteer) to the general list of signs and acronyms specified in general_currency.

```
en_monetary_amount  :>
         @seek(general_monetary_amount)
        (
        (morph & [ STEM    "million" ,   ID #magnitude & "10e6" ]
        | token & [ SURFACE "m",    ID #magnitude & "10e6" ]
        | morph & [ STEM    "billion" ,   ID #magnitude & "10e12" ])
        ) ?
         -> dummy.

en_currency :>
               Gazetteer & [TYPE en_currency, SURFACE #currency]
               |
               @seek(general_currency)
        -> dummy.
```

By using these two rules, which include general rules, en_money below makes sure that complete monetary expressions combining both general and language specific tokens (e.g. "$300 million", or "300 million dollars") are matched.

```
en_money    :>
               ( @seek(en_currency)
                 @seek(en_monetary_amount) )
         |
        ( @seek(en_monetary_amount)
                 @seek(en_currency)  )
               ->
               money & [ CURRENCY  #currency ,
                              AMOUNT    #amount ,
                     MAGNITUDE #magnitude ].
```

Parallel structures to en_monetary_amount, en_currency and en_money had been previously written for French and Spanish, so the work involved in generating the English rules came down to copying those rules into a new file, changing the words for "million" and "billion" and providing the Gazetteer list for English currencies.

This procedure has not only been used for generic vs. language-specific grammars, but also for the development of general vs. application-specific grammars. Thus, the grammars for general NER were used for the European project AIRFORCE (AIR FOReCast in Europe)[3], whose aim is to evaluate the contribution of advanced statistical methods, combining intelligent agents and data mining algorithms to forecast the number of air passengers for various destinations in Europe. Text mining for AIRFORCE involved extracting only certain named entities, and all we had to do was implement the new application-specific rules on top of the already existing ones. As an example, our general grammars recognized all types of dates, but in the new application we only wanted to get some of the dates in a given text, namely those that referred to events for which we had information on the number of visitors they attracted. What we did was add rules constraining the contexts of the dates we were interested in. The following examples illustrate this.

The first rule below (en_deictic_year) defines a key element covering date structures such as "last year", "past year" or "next year". The first rule is embedded in the second rule en_airforce_visitors_year, which uses yet another key element en_visitors_number (covering structures like e.g. "2000 visitors"). The second rule would cover a sentence such as "Last year the event attracted 2000 visitors".

```
en_deictic_year :>
  (morph & [STEM "this" , ID "this" & #year_id]
    token & [SURFACE "year" ] )
  |
  (
   (  morph & [STEM "last" , ID "last" & #year_id]
    | morph & [STEM "past" , ID "last" & #year_id]
   )
   token & [SURFACE "year"]
  )
  |
  ( morph & [STEM "next" , ID "next" & #year_id]
    token & [SURFACE "year"]
  )

-> dummy.
```

[3] For more information, please see the official page of the project: www.sofresud.com/airforce/

```
en_airforce_visitors_year :>
  @seek(en_deictic_year)
  token*
  morph & [STEM "attract"])
  @seek(en_visitors_number)
-> statistics & [  VISITORS  #visitorsNumber ,  YEAR  #year_id].
```

The new rules that were written for the specific application could be added in different files, so that the general rules for dates did not need to be modified at all.

The division into shared generic and language-specific parts has also been applied in the gazetteer, a component which is used to store lists of locations, companies, names, etc. Thus, names that are common to the three languages, e.g., "Amsterdam" have been defined in generic lists, while language-specific denominations such as "Brussels", "Bruxelles" and "Bruselas" are stored in the corresponding language-specific lists. This helps reducing the size of lists, which is important especially in the domain of NE-grammar development, where this storage of information is heavily used.

The development of multilingual parallel grammars as described above has turned out to be very efficient, since by increasing modularity, it also increases reusability and extensibility.

4 Multilingual Named Entity Corpora

We use corpora annotated with named entities for grammar development, and for evaluation of the grammars with respect to recall, precision, and F-measure. Ideally, the markup used in the annotated corpora would correspond exactly to the output produced by the grammar, so that mismatches could be detected easily. Such an ideal case is possible in large-scale evaluations, such as MUC, but is likely to exceed the resources of most application-oriented projects, which have to re-use existing corpora even if they do not fit perfectly with their objectives.

In our case, we have made use of the following corpora annotated with NEs.

- English corpora from the MUC7 evaluations
- Japanese and Chinese corpora annotated according to MUC7 conventions
- German corpora annotated in the COLLATE project with a superset of MUC7 annotations[4]
- German, English, French and Spanish texts annotated with Named Entities, from JRC
- Spanish data from the CoNLL-2002 Language-Independent NER task
- English and French corpora from the business domain annotated with named entities according to the MUC7 guidelines within our project

In some cases, it makes sense to use multiple corpora for the same language due to different and complementary coverage of each corpus. The different corpora may be very rich or poor with respect to particular kinds of named entities (e.g. company names are very frequent in business news), and may cover different domains and genres. In particular, it may make sense to have some corpora with development and test data for language-independent NER, complemented by domain-specific corpora for particular applications such as business or sports.

While our multilingual grammars make produce the same language-independent output structures for each language, the annotated corpora contain differences

- in annotation format,
- in the types of named entities annotated, and
- in the attributes used to describe each NE.

The first kind of difference, superficial syntactic differences, can be dealt with by a transformation to a common (XML-based) format, while the other two kinds of differences are more serious. We do not address them by modifying or re-annotating the corpus, but rather by using a flexible evaluation tool, which can deal with differences between the output structures produced by a NE grammar and the NE annotations in the evaluation corpus, as described in the following section.

By following this approach, we can achieve maximal re-use of existing corpora, and can limit labor-intensive corpus annotation to cases where the coverage of the existing corpora with respect to language, domain or genre is insufficient.

[4] This German corpus has been annotated on multiple linguistic layers (Named Entity, domain templates, coreference), and is being used for other purposes besides NER (Callmeier et al., 2002)

5 Evaluation Tool

We have developed a tool (jTaCo) for the evaluation of grammars with respect to a corpus annotated with corresponding structures. The tools works by removing any annotations from the corpus, and feeding the unannotated texts to the grammar. It then compares the grammar output to the original annotated files, and produces detailed statistics, evaluation scores, and diagnostic output for helping the grammar writer by highlighting differences between the grammar output and the annotated corpus. The architecture of jTaCO is shown in figure 4.

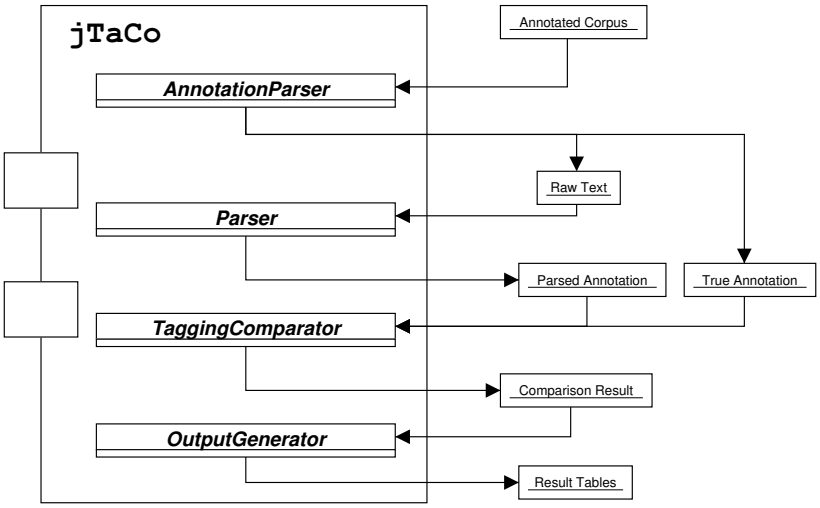


Figure 4.Architecture of jTaCO

jTaCo can be configured to deal with various problems in evaluating grammars with respect to a corpus:

- use of different classes of named entities, or different granularities (e.g., *organization* and its subclasses *company, university, government etc.*)
- the extent of a NE may be different: e.g., a person name may or may not include a function and a title ("Chairman and CEO Bill Gates" vs. "Bill Gates")
- the markup of the corpus may be textually oriented (e.g., as XML tags) while output of grammar is a semantic structure

For the first problem, jTaCo allows the user to declare the equivalence of classes, and subclass relationships.

For the second problem, jTaCo can be configured to accept particular named entities as recognized correctly even if the left and right boundaries are not exactly the same as in the corpus. The user can specify that only the left or right boundary must be matched, that the recognized NE must be included in the annotated one or vice versa, or that the NEs must overlap, but not match exactly. The size of the allowable mismatch on each side (number of tokens) can be specified by the user. These options can be specified separately for each NE class, and for each input corpus.

The third problem arises from the fact that manual corpus annotation is generally additive, i.e., the original corpus is preserved and annotations are added to it, while NLP applications such as NER are often transformative, i.e., the original input is being replaced by higher-level representations (tokens, morphological, syntactic, semantic). SProUT produces typed feature structures as output, but stores information in the output structure about the origin of each output structure with respect to tokens in the input. This correspondence between the semantic output structure and the input from the test corpus is used for evaluating the output structure against the corpus. In case of a semantic output structure, the problem of partial recognition of NPs is not matching of string boundaries, but matching of semantic representations against a named entity annotated in the corpus.

The solutions to the first and second problem are fully implemented and configurable in a graphical user interface. The solution to the third problem is still being implemented.

6 Conclusion and Future Work

We have addressed a fundamental problem in re-using heterogeneously annotated corpora as training and testing data for multilingual grammar development, and have outlined our approach to the problem.

With the increasing availability of annotated corpora on the web or through distribution agencies, such as ELRA or LCD, the re-use of annotated corpora becomes an attractive and cost-effective option, compared to the special-purpose annotation efforts. In this work, we have described methods and tools for re-using annotated corpora for the development and evaluation of NE grammars.

7 Acknowledgements

The research underlying this paper was supported by research grants from the German *Bundesministerium für Bildung, Wissenschaft, Forschung und Technologie* (Projects WHITEBOARD, 01 IW 002, and COLLATE, 01 IN A01), and from the European Commission (Project AIRFORCE, IST-12179).

8 References

[Asahara, 2000] M. Asahara and Y. Matsumoto. *Extended Models and Tools for High-Performance Part-of-Speech Tagger*, In Proceedings of the 18th COLING, pages 21-27, 2000.
[Becker et. al, 2002] M. Becker, W. Drożdżyński, H.U. Krieger, J. Piskorski, U. Schäfer, F. Xu. *SProUT - Shallow Processing with Typed Feature Structures and Unification*. In Proceedings of ICON 2002 - International Conference on NLP, Mumbai, India, December, 2002.
[Chinchor 1997] Nancy Chinchor. *MUC7 Named Entity Task Definition.* Technical Report, NIST, September 1997.
[Callmeier et al. 2002] U. Callmeier, G. Erbach, I. Gogelgans, S. Hansen, K. Kunz and D. Ziegler-Eisele. *COLLATE Annotationsschema.* Technical Report, Saarland University, 2002.
[Krieger and Schäfer, 1994] H.-U. Krieger, U. Schäfer. *TDL – A Type Description Language for Constraint-Based Grammars*. In Proceedings of COLING, pages 893-899, 1994.
[Liu, 2001] K. Liu. *Research of automatic Chinese word segmentation*. In International Workshop o n Innovative Language Technology and Chinese Information Processing (ILT&CIP-2001), 2001.
[Petitpierre and Russell, 1995] D. Petitpierre and G. Russell. *MMORPH-The Multext Morphology Program*, 1995. Multext deliverable report 2.3.1. ISSCO, University of Geneva.
[Piskorski et. al, 2002] J. Piskorski, W. Drożdżyński, F. Xu, O. Scherf. *A Flexible XML-based Regular Compiler for Creation and Converting Linguistic Resources*. In Proceedings of LREC 2002
[Piskorski, 2002] J. Piskorski. *DFKI Finite-State Machine Toolkit*. Research Report RR-02-04, DFKI GmbH - German Research Center for Artificial Intelligence, Saarbrücken, Germany, 2002.