A FRAMEWORK FOR DOMAIN AND TASK ADAPTIVE NAMED–ENTITY RECOGNITION

Jakub Piskorski
DFKI GmbH
Stuhlsatzenhausweg 3, 66123 Saarbrücken
piskorsk@dfki.de

Tilman Jäger
XtraMind
Stuhlsatzenhausweg 3, 66123 Saarbrücken
jaeger@xtramind.com

Feiyu Xu
DFKI GmbH
Stuhlsatzenhausweg 3, 66123, Saarbrücken
feiyu@dfki.de

Abstract Robust Named–Entity Recognition software is an essential preprocessing tool for performing more complex text processing tasks in business information systems. In this paper we present a Framework for Domain and Task Adaptive Named–Entity Recognition. It consists of several clear–cut subcomponents which can be flexibly and variably combined together in order to construct a task–specific NE–Recognition tool. Additionally, a diagnostic tool for automatic prediction of best system configuration is provided, which speeds up the development cycle.

1. Introduction

Nowadays, knowledge relevant to business of any kind is mainly transmitted through free–text documents. New trends in information technology such as *Information Extraction, Text Mining* or *Text Classification* could provide dramatic improvements in the process of converting the overflow of raw textual information into valuable knowledge. Since named entities (NE) constitute a

significant part of business texts[1] , robust NE–Recognition software is an essential preprocessing tool for performing more complex text processing tasks in business information systems (Abramowicz and Zurada, 2001).

The NE task has formally been defined at the MUC conference (SAIC, 1998) and consists of recognition of *entities* (organizations, persons, locations), *temporal expressions* (date, time) and *quantities* (monetary values, percentages). The recognition process is usually subdivided into delimitation, i.e. determination of the boundaries of the NE, and classification, in which the identified NE is assigned a more specific category. The question whether a phrase is a named entity and what name class it belongs to, might depend on both internal structure and surrounding context (McDonald, 1993). The ”Inc.” designator reliably shows the phrase ”Financial Investments, Inc.” to be a company name. The text fragment ”Paco Raban” in the following example sentence can be recognized as a company name by utilizing the preceding word sequence ”director of”.

(1) Mr. Diagne would leave his job as vice–president of Yves Saint Laurent, Inc. to become operations director of Paco Raban.

However, ”Paco Raban” could also be a person name. In this case, the problem of disambiguating the type of this NE seems to be intuitively simple, but generally, a broader contextual knowledge is required in order to determine the type of the named entity correctly. An intuitive strategy for NE recognition one could think of is simply using dictionaries containing NEs. However, some NEs, like for instance company names are too numerous to include them in dictionaries. Further, they are changing constantly and may appear in many variant forms (e.g. subsequent occurrences might be abbreviated). Hence, such straightforward list–search approach would not perform well.

Most of the IE systems participating in the NE task in the recent MUC conferences were based on handcrafted approach (Wakao et al., 1996), which use context–sensitive rules. Such contextual rules rely usually on tokenization and/or lexical information computed in the preprocessing phase. Recently, a variety of machine learning methods for solving NE task were introduced (Bikel and Weischedel, 1997, Borthwick, 1999, Gallippi, 1996, Srihari et al., 1999). Nevertheless, the handcrafted systems achieve on an average higher coverage than machine–learning based approaches. This is due to the fact that non–local phenomena are best handled by using regular expressions. For instance, the pattern [PERSON_NAME ∘ STRING:THE ∘ (POS:ADJ)* ∘ STRING:PRESIDENT OF ∘ ORG_NAME], which covers the following example phrase would be difficult to learn for an automated system because of the presence of a sequence of zero or more adjectives:

[1]According to (Coates-Stephens, 1992) newswire texts may consist of up to 10 percent of proper names.

(2) Bill Gates, the young dynamic successful president of Microsoft

A broader overview of existing NE recognizers and current problems in this field are given in (Borthwick, 1999).

In this paper we describe a Domain and Task Adaptive Named–Entity Recognition Platform. In contrary to other tools, our platform comprises of several clear–cut submodules which can be flexibly and variably combined together in order to construct a task–specific NE–recognition component. The submodules include among others fine–grained tokenization, regular pattern matching, acronym recognition, gazetteer lookup, NE variant identification and lexical processing. The main advantage of the flexible workflow is that unnecessary computations might be avoided (e.g., specific IE tasks do not require lexical processing) and the best configuration for solving a particular real–world extraction task may be easily defined. Therefore, an additional diagnostic tool for automatic evaluation and scoring of possible system configurations is provided, which means faster development cycle. Since it is important to process big quantities of texts efficiently, we make an exhaustive use of *finite–state technology* (Mohri, 1996) in all subcomponents[2]. Currently, we have adapted the system for performing NE–Recognition for German. Additionally, we highlight frequent problems we encountered while processing real–world data, which illustrate the indispensability of each system component.

2. Architecture

The system consists of two main pools of resources: (1) *the linguistic resources*, which are maintained (and optimized) by the *Grammar Manager*, and (2) *processing resources*, including Tokenizer, Pattern Matcher, Gazetteer Checker, Context Explorer, Lexical Processor and Acronym Finder. The processing resources can be flexibly and variably combined into a task–specific workflow schedule through the use of the *Resource Scheduler*. The workflow schedule determines which components of the system are used and defines the order of their application and interplay.

Each text document is firstly preprocessed by the tokenizer which is an obligatory step in our NE–recognition system. The *Tokenizer* splits the text into word–like units called tokens and classifies them according to user–defined token–class definitions. The segmented text together with the workflow schedule constitute the input for the *NE–Recognition Engine* which performs the proper NE identification. The text fragments which are recognized by a certain module as NEs are by default not further consumed by the subsequent mod-

[2]Computationally, finite–state devices are time and space efficient and from the linguistic point of view, local natural language phenomena can be easily and intuitively expressed as finite–state devices.

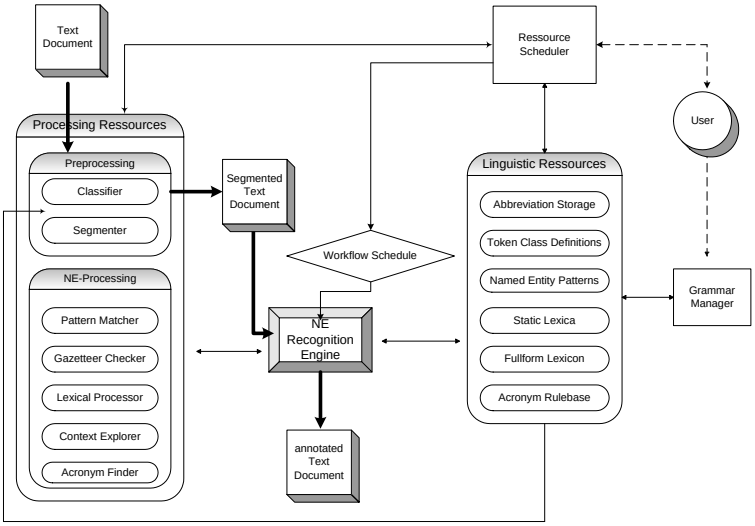

Figure 1. Architecture

ules[3]. The extracted information may be redirected into a corresponding XML document or stream of feature structures.

We describe now briefly the functionality of the processing resources. The *Pattern Matcher* applies handcrafted rules representing patterns for NE recognition. The patterns rely on tokenization information and information computed by the *Lexical Processor* which is responsible for retrieving lexical information (including online compound recognition) for each unconsumed token. The task of *Acronym Finder* is the recognition of acronyms and identification of their corresponding definitions. The *Gazetteer Checker* uses static NE–lexica (e.g., geographical locations) in order to match unconsumed text fragments. Finally, the *Context Explorer* searches for associations between token sequences which do not exhibit strong named entity evidence (potential NE) with named entities which were recognized within a parametrizable context frame (e.g., different variants of the same NE).

A scenario–specific workflow schedule may include more than one instance of some of the processing resources. For instance, in some scenario the set of recognition rules for the Pattern Matcher could be subdivided into sure–fire

[3]More complex configuration of the system allow for consuming previously recognized entities which will be discussed later.

rules and less–reliable rules (e.g. rules obtained statistically by some external component, which can be easily converted into an appropriate format). In this case it would be convenient to firstly apply sure–fire rules, then to run the Gazetteer Matcher and finally to use the set of less–reliable rules as argued in (Mikheev et al., 1999). For speeding up the adaptation to new scenarios the best system configuration can be automatically predicted by a diagnostic tool provided in our system.

Finally, in order to guarantee good run–time performance all components of the systems are implemented as optimized finite–state devices. We used the DFKI Finite–State Machine Toolkit (Piskorski, 2002) for constructing, combining and optimizing finite–state machines which are generalizations of weighted finite–state automata and weighted finite–state transducers.

3. Processing Resources

3.1 Tokenizer

The task of the tokenizer is to map character sequences of the input text documents into word–like units called tokens and to classify those tokens according to user–defined token classes. In the first step, the *Segmenter* splits a text document into tokens by using an abbreviation lexicon which allows for a rough distinction between points being integral part of the token (e.g. abbreviations) and sentence delimiters. The segmented text is forwarded to the *Classifier*, which performs fine–grained categorization of tokens. In contrary to other tokenization tools our tokenizer allows for multiple token classification, as illustrated in figure 2. Firstly, each token is classified according to a prespecified list of main token classes (currently the system provides about 50 default main token classes).

Example Tokens	Maintoken Classes	Subtoken Classes
1999	natural number	four digit number, year
Düsseldorf	first capital word	potential location
V3.5	number word compound	–
AT&T–CEO	complex compositum	may include company name
GmbH	mixed word first capital	corporate designator

Figure 2. Token classes

In the second step, tokens undergo additional domain and language specific subclassification (Thielen, 1995). For instance, we could subclassify all first–capital tokens according to the type of suffix they end with (e.g. suffix exhibiting a potential location: ”–dorf”, ”–hafen” in German). Such information could be used for defining of NE–recognition patterns (e.g. location prepositions followed by token subclassified as potential location). Each token may

belong to several subtoken classes, but it is assigned to only one main token class. All token classes expressed as regular expressions are merged into a single optimized finite–state network which represents the token classifier. We claim that fine–grained tokenization as described here may be sufficient for solving some real–world NE tasks as we experienced in practice (e.g. keyword–spotting).

3.2 Pattern Matcher

The Pattern Matcher applies handcrafted regular patterns (Mikheev et al., 1999, Gallippi, 1996), which rely on the tokenization information and information computed by the lexical processor (see section 3.4). These patterns are expressed as finite–state devices whose arcs are labeled with predicates on tokens. There are only five types of predicates (STRING, STEM, POS, SUBTOKEN AND TOKEN) which allows for keeping the degree of non–determinicity of the corresponding finite–state grammar relatively low. An example pattern for recognition of company names is given in figure 3:

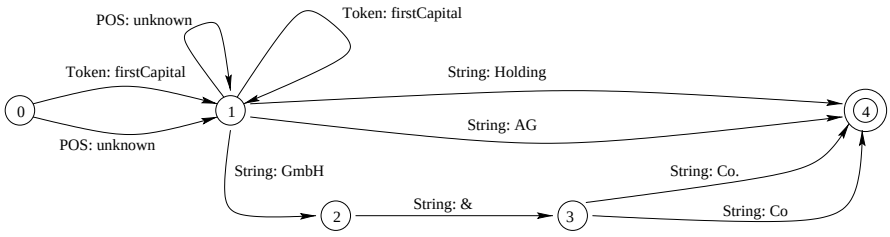

Figure 3. A simple pattern for recognition of company names

The Pattern Matcher can be configured in several ways, which can influence the coverage and runtime behavior of the system. Firstly, various matching techniques can be chosen (e.g. local vs. full backtracking). Another option allows for recognizing overlapping NEs. Consider the following example:

(3) Bei XtraMind Technologies GmbH, Stuhlsatzenhausweg 366121 Saarbrücken

In this text fragment two overlapping NEs could be identified: a location ”Bei XtraMind” triggered by a location preposition, and ”XtraMind Technologies GmbH” triggered by the company designator ”GmbH”. Such collision information could be used for semi–automatic fine–tuning of the rule–base (definition of new pattern covering the whole text fragment ”Bei XtraMind ... GmbH”). Further, the above text fragment contains an error (e.g., originating from OCR), in that it misses a space between street number of

"Stuhlsatzenhausweg" and city code of Saarbrücken (fusion of 3 and 66121). Using collision–matching option we can identify both street–number fragment and postcode–city phrase which would not be possible otherwise. Additionally, an option for determining the order and number of admissible predicates is provided, which may be used for optimizing the Pattern Matcher in terms of efficiency. Pattern Matcher also allows for rule prioritization which prevents multiple assignment to same token sequences. Finally, on demand previously recognized NEs can be consumed by the Pattern Matcher, e.g. person names recognized by Gazetteer Checker. Note that an input text can be partially annotated.

3.3 Gazetteer Checker

The task of the Gazetteer Checker is recognition of NEs stored in static NE–lexica. Such lexica contain usually location, organization and person names. Simple NE–Recognition systems rely only on performing a lexicon lookup. The advantage of using gazetteers may be verified by the fact that many NEs do neither exhibit internal nor external evidence of being a named entity. For instance, consider the German phrase "Dynamik in Handel" (*dynamic in trade*). It is a magazine title, but also a valid NP in German. Using Gazetteer seems to be an only alternative for recognizing such named entities. An interesting discussion of the importance of using gazetteer and their application at different stages of NE–recognition process can be found in (Mikheev et al., 1999). In case of ambiguous entries in static lexica, we can switch between an option of returning the highest–priority interpretation or returning all possible interpretations. For converting the static lexica into their corresponding optimized finite–state representation we use the new method for efficient incremental construction of acyclic deterministic and minimal finite–state automata presented in (Daciuk, 1998) and provided by the FSM Toolkit.

3.4 Lexical Processor

The task of Lexical Processor is the retrieval of lexical information for each token identified as potential word form. This includes also recognition of compounds (e.g. "Produktionsumstellungen" – *production reorganization*) which are usually not lexicalized [4]. They constitute significant part of business texts [5]. Lexical Processor uses full-form lexicon (750 000 entries for German) and tries firstly to associate each processed token with a corresponding lexical information including part–of–speech and stem information. If no such information

[4]Note that, for instance in German compounds are in general orthographically single words.

[5]We found out that 7,19 percent of the words in our test corpus, consisting of business articles from the German business news magazine "Wirtschaftswoche", were compounds.

can be found, the token is either compound or unknown word. We apply fast and robust shallow compound recognition strategy outlined in (Piskorski and Neumann, 2000) which computes only a single syntactically valid segmentation and determines the head while leaving internal bracketing underspecified. This information is sufficient for the purpose of NE recognition. The stand–alone status of the Lexical Processor may be verified by the fact that one aims to applying other components to text fragments consisting of tokens not recognized as valid word forms. Analogously to static lexica we encode full–form lexicon into an optimized finite-state network.

3.5 Context Explorer

The task of the Context Explorer is the identification of variants of already recognized NEs. Hence, this component fulfills partial coreference resolution. The text including annotations of previously recognized entities is scanned in order to identify candidates for NEs (which do not exhibit a strong evidence of being a named entity). Secondly, Context Explorer searches for associations between such candidates and already identified NEs within a parametrizable context frame. For instance, first occurence of a given company name usually includes a designator, whereas subsequent occurrences are frequently abbreviated variants, which do not include such designators and are thus harder to find (e.g. "Appollinaris & Schweppes" and "Appollinaris" could refer to the company name "Appollinaris & Schweppes GmbH & Co."). This is achieved by storing certain types of NEs (e.g. company name without corporate designator) in a dynamic lexicon. Any subsequent occurrence of a prefix or suffix of previously stored NEs can then be identified and an appropriate reference can be set correctly. It is known that such reference resolution heuristic achieves high accuracy.

Another advantage of using dynamic lexicon is that it can be used for disambiguating NE types. Consider the text in example 4:

(4) "Ich könnte niemals auf irgend etwas schiessen", versichert der 57jährige Chef des US–Rüstungskonzerns Martin Marietta Corp. (MM). Doch die private Waffenabstinenz hat Augustine nicht daran gehindert, sein Unternehmen zur grössten Waffenschmiede der Welt aufzurüsten: Für drei Milliarden Dollar hat Martin Marietta gerade erst die Luftfahrtabteilung des ehemaligen Konkurrenten General Electric (GE) übernommen und damit seinen Jahresumsatz von rund sechs auf über elf Milliarden Dollar fast verdoppelt."

The first occurrence of the sequence "Martin Marietta Corp." can be easily identified as a company name, since it contains a reliable corporate designator. The subsequent occurrence of the substring "Martin Marietta" could be firstly

recognized by the Pattern Matcher or the Gazetteer Checker as a person name. Nevertheless, this token sequence obviously refers to the company name introduced previously. A simple heuristic may be applied to solve this problem: whenever a person, company or location name is recognized by the Pattern Matcher or Gazetteer Checker, the Context Explorer performs an additional lookup in order to check whether such NE is a prefix or suffix of an already recognized named entity in the surrounding context frame, and modifies the type of this NE appropriately.

The Context Explorers' ability for varying the size of the context window (e.g. paragraph vs. the entire document) is crucial since the system has to cope with different types of documents (e.g. emails, web pages or newswire articles).

3.6 Acronym Finder

We noticed by looking over a lot of business documents that they include a huge number of acronyms which can be treated in most cases as NEs (e.g. "GE" in example 4 stands for "General Electric"). In order to recognize acronyms and their corresponding definitions we apply two strategies: (1) by the definition of lexico–syntactic patterns expressing introduction and definition of an acronym (e.g. X steht für Y - *X stands for Y*), (2) by applying rules for identifying acronym candidates and associating these candidates with their definitions in an parametrizable context frame using heuristics as described in (Roy, 2001). The two strategies can be applied simultaneously or separately. With regard to (2) a more detailed description follows:

By taking advantage of the information computed by the Tokenizer it is possible to find short character sequences (e.g. the length of a sequence lies between 2 and 10 characters, tokens consisting solely of consonants) which constitute *acronym candidates.* For each acronym candidate an abbreviation pattern reflecting its structure is constructed. The pattern consists of a sequence of symbols which correspond to the subsequent characters in the acronym candidate ("n" stands for number and "c" stands for character). Some characters like "&" or "\" have no corresponding symbol in the pattern. Figure 4 shows some example acronyms and their corresponding patterns:

Acronym	Pattern
AT&T	ccc
2D	nc
T/C/F	ccc
ALGOL	ccccc

Figure 4. Acronym Patterns

Definition	Pattern
Anglo–Australien Observatory	www
USA	wwsw
ALGOrithmic Language	hw

Figure 5. Definition Patterns

Subsequently, Acronym Finder tries to identify *definition candidates* in the surrounding context of the acronym candidate using several heuristics. Consider again the example 4, where ”GE” stands for ”General Electric”. An appropriate heuristic for recognizing the definition candidate would be as follows: initial character of each word of the definition candidate must match the corresponding character in the acronym candidate. In the next step a *definition pattern* representing the structure of the definition candidate is built. Some examples of acronym definitions with their corresponding definition patterns are given in figure 5 (”w”- words initialized with a capital, ”h”- words containing two or more capitals and ”s”- stopwords). Analogously to abbreviation patterns, some separator symbols are ignored.

After the identification of acronym and definition candidates and generation of their corresponding patterns, the Acronym Finder compares the pattern pair with pattern pairs stored in the *Acronym Rulebase*, and returns one or more corresponding Formation Rules. *Formation Rules* express the way how an acronym is formed from its definition. Finally the returned set of formation rules is applied to definition candidate in order to validate the acronym candidate. The formation rule $< wwsw, ccc, (1, f)(2, f)(4, f) >$ can be interpreted in the following way: take initial letter of the first, second and fourth word in the definition and omit the stopword (third word). In this way the acronym ”USA” is constructed from the phrase ”United States of America”. Currently, the Acronym Rulebase contains a collection of ca. 50 handcrafted formation rules, which can be extended by machine–learning techniques (Roy, 2001). Note that already recognized NEs occurring in an immediate context of acronym candidates are used as preferred definition candidates.

4. Diagnostic Tools

In order to reduce the time for adaptation of the system to new scenarios, a diagnostic tool for automatic prediction of best system configuration is provided. The optimal configuration respectively precision, recall or f–measure can be computed by means of an annotated text corpus provided by the user, and a set of candidate configurations (workflow schedules). A configuration candidate is an ordered list of instances of processing resources. The set of configurations can be automatically computed, which is done in the following way. Firstly, the user selects available linguistic resources, e.g. static NE–lexica, pattern sets, etc. Subsequently, the system generates a set of all corresponding instances of processing resources. Finally, for each element of the power set of this set, all possible permutations are computed. These permutations constitute configuration candidates.

Since the number of all permutations for a larger set of instances of processing resources is obviously too numerous to be computed efficiently, sev-

eral options may be used for reducing the number of configuration candidates to be considered. Firstly, the user may restrict the number of potential candidates by defining partial linear precedence constraints (e.g. ”component X cannot be used before component Y”) or by using other filters such as: ”consider only configuration candidates consisting of three components”. Besides the option of computing all permutations, several efficient heuristics for predicting best/near–optimal configurations may be applied. For instance, in the ”bottom–up” heuristic, we firstly evaluate all configurations consisting of single components and use *k* top scoring components wrt. the chosen measure in order to generate configuration candidates consisting of two components. Further, we evaluate the two–component configurations and proceed in the same way until configurations consisting of maximal number of components have been evaluated. For avoiding unnecessary recomputations we use cashing–techniques.

Furthermore, the user may also specify his preferred configurations and edit the automatically created list of candidate configurations for fine–tuning the evaluation process. The best configurations are presented to the user in form of ranked lists according to the chosen evaluation measure, where the the evaluation process can be done in two fashions: exact vs. partial matching (i.e. left or right boundary of the recognized entity matches with the entity annotated in the test corpus).

For the sake of clarity we summarize the configuration options of all components in the table of figure 6. The following example scenario illustrates the

Configuration Options		
Components	*Multiple Instances*	*Main Configuration Options*
Tokenizer	yes	Language and domain adaptive token subclassification
Lexical Processor	no	no
Context Explorer	yes	Various size of context window
		NE–type disambiguation
Gazetteer Checker	yes	Filters for selecting text passages to be analyzed
		Lexicon Prioritization
Acronym Finder	yes	2 acronym recognition strategies
Pattern Matcher	yes	Rule Prioritization
		Various backtracking options
		Longest match vs. collision match
		Predicate order and number
		Consuming already identified NEs

Figure 6. Overview of the Configuration Options

importance of the diagnostic feature of the system. Let us assume that we are interested in the recognition of company and person names. The set of linguistic resources contains of gazetteers for first and last names, and a set of patterns for recognition of person and company names. Let *A* denote the workflow schedule in which we firstly apply Gazetteer Checker and subsequently the Pattern Matcher; and let *B* denote the workflow in which the above processing resources are applied in reverted order. Assuming that we use the pattern [TOKEN:PERSON ∘ STRING:& ∘ TOKEN:PERSON] for recognition of company names (which is obviously more reliable than the pattern [TOKEN:FIRST_CAPITAL ∘ STRING:& ∘ TOKEN:FIRST_CAPITAL], the workflow schedule *A* seems to be superior to *B* in order to identify "Alexander & Alexander" as a company name.

Now, let us consider the following phrase: "Der Müller Klaus Kinski" – *the miller Klaus Kinski*, where Müller is a valid last name in German. For recognizing person names we would expect to achieve good coverage by applying following patterns: (1) [(POS:UNKNOWN)$^+$∘ TOKEN:ENDS_WITH_SUFFIX_SKI], (2) [TOKEN:LAST_NAME∘TOKEN:FIRST_NAME] and (3) [TOKEN:FIRST_NAME ∘TOKEN:LAST_NAME]. *B* is superior to *A* in order to identify the text fragment "Klaus Kinski" as person name correctly (only pattern (1) matches). If the application of Gazetteer Checker precedes the application of Pattern Matcher (workflow schedule *A*) two colliding person names would be identified ("Müller Klaus" – pattern (3) vs. "Klaus Kinski" – pattern(2) and (1)). Note that it would be only possible in the collision match modus of the Pattern Matcher. This example proves that both workflow schedules exhibit advantages and disadvantages for recognition of different types of NEs, which validates the relevance of the diagnostic tool.

Further, the list of top–scoring configurations might give an insight into the contribution and usefulness of the particular processing and linguistic resources being used. In this way, time and space expensive resources which do not significantly contribute for achieving good precision and recall values, can be eliminated. In particular, the information concerning best configurations for different text fragments allows for merging different workflow schedules. The figure 7 shows precision and recall values for some workflow schedules dedicated for company name detection. For solving this task, we used a gazetteer for company names (COMP–G), containing 3271 entries, and a set of ca. 30 recognition patterns (COMP–PM). Further, we allowed to use Context Explorer (CE) and Acronym Finder (AF). For the evaluation we used a test corpus, consisting of business news articles from the German business news magazine "Wirtschaftswoche" (200 000 tokens).

Evaluation of some Workflow Candidates		
Workflow Candidates	*Precision*	*Recall*
COMP–G	89,2%	24,5%
COMP–PM	93,9%	76,33%
COMP–G CE	83,83%	30,1%
COMP–G CE COMP–PM	95,19%	86,76%
COMP–G CE COMP–PM AF	95,1%	87,09%
COMP–PM COMP–G CE AF	94,4%	86,5%

Figure 7. Evaluation

5. Summary

Robust NE–recognition is prerequisite for successfully performing other more complex extraction and mining tasks in the context of business information systems. Most of the recent research in the area of extracting named entities from free texts centers around systems tailored to a specific domain. This paper describes a Domain and Task adaptive Named Entity recognition framework. It consists of several clear–cut subcomponents which can be flexibly and variably combined together in order to construct a task–specific NE–recognition engine. Further, it provides a diagnostic tool which aims to automatically predict optimal configuration of the system (e.g., which processing and linguistic resources to use) in a given scenario. In this way the one can speed up the time–consuming adaptation and optimization process. The system has been mainly implemented in JAVA, except the Finite–State Toolkit, which has been implemented in C++. To our knowledge no similar NE–Recognition tools have been described in the literature, and therefore it is difficult to compare the presented framework with other approaches.

Currently, we continue testing the system on texts drawn from the financial domain using larger amount of instances of processing resources in order to evaluate the configurability features. Future work will focus on integration of

trainable agents based on machine learning techniques for discovering NE–recognition patterns (Bikel and Weischedel, 1997), which could be applied when sufficient training data is provided. Further, we intend to investigate the usability of integrating an ontology component (e.g. for synonym recognition). Finally, additional work will be spent on improving the techniques for approximation of best system configuration.

References

Abramowicz, W. and Zurada, J., editors (2001). *Knowledge Discovery for Business Information Systems*. Kluwer Academic Publishers.

Bikel, Miller, S. and Weischedel, N. (1997). A high-performance learning name-finder. In *Proceedings of ANLP, Washington, DC, pages 195-201*.

Borthwick, A. (September, 1999). *A Maximum Entropy Approach to Named Entity Recognition*. PhD thesis, New York University.

Coates-Stephens (1992). *The Analysis and Acquisition of Proper Names for Robust TextUnderstanding*. PhD thesis, Department of Computer Science, City University London.

Daciuk, J. (1998). *Incremental Construction of Finite–State Automata and Transducers and their use in the Natural Language Processing*. PhD thesis, University of Gdansk, Poland.

Gallippi, A. F. (1996). A synopsis of learning to recognize names across languages. In *Proceedings of ACL*.

McDonald, D. . (1993). Internal and external evidence in the identification and semantic categorization of proper names. In *Proceedings of the SINGLEX workshop on "Acquisition of Lexical Knowledge from Text"*.

Mikheev, A., Moens, M., and Grover, C. (1999). Named entity recognition without gazetters. In *Proceedings of the ninth international Conference of the European Chapter of the Association for Computational Linguistics (EACL 99)*.

Mohri, M. (1996). On some applications of finite-state automata theory to natural language processing. *Natural Language Engineering Vol. 2 No. 1*, 2(1):61–80.

Piskorski, J. (2002). Dfki finite state machine toolkit. Technical report, DFKI GmbH, Saarbrücken, Germany.

Piskorski, J. and Neumann, G. (2000). An intelligent text extraction and navigation system. In *Proceedings of the 6th International Conference on Computer Assisted Information Retrieval (RIAO–2000)*, Paris.

Roy, Y. P. (2001). Hybrid text mining for finding abbreviations and their definitions. In *Proceedings of NACL*.

SAIC, editor (1998). *Seventh Message Understanding Conference (MUC-7)*. `http://www.muc.saic.com`.

Srihari, R. K., Srikanth, M., Niu, C., and Li, W. (1999). Use of maximum entropy in back-off modeling for a named entity tagger. In *Proceedings of the HKK Conference*, Canada.

Thielen, C. (1995). An approach to proper name tagging in german. In *Proceedings EACLSIGDAT 95*.

Wakao, T., Gaizauskas, R., and Wilks, Y. (1996). Evaluation of an algorithm for the recognition and classification of proper names. In *Proceedings of the 16th International Conference on Computational Linguistics (Coling96), Copenhagen*.