УЧЕНЫЕ ЗАПИСКИ КАЗАНСКОГО УНИВЕРСИТЕТА
Том 155, кн. 4 Физико-математические науки *2013*

UDK 004.912

SEMI-AUTOMATIC GENERATION OF LINEAR EVENT EXTRACTION PATTERNS FOR FREE TEXTS

D. Dzendzik, S. Serebryakov

Abstract

In this paper we describe semi-automatic approach to generating event extraction patterns for free texts. The algorithm is composed of four steps: we automatically extract possible events from a corpus of free documents, cluster them using dependency-based parse tree paths, validate random samples from each cluster and generate linear patterns using positive event clusters. We compare it with the system that uses handcrafted patterns.

Keywords: event extraction, linear patterns, regular expressions, TextMARKER, RUTA.

Introduction

The event extraction (EE) problem is formulated as "to automatically identify events in free texts and derive detailed information about them, ideally identifying who did what to whom, when, with what methods, where and why". Events involve entities and relations between them and imply a change of state. For instance, the sentence "Palm has been acquired by Hewlett-Packard for $1.2 billion two days ago" mentions the event of type *Mergers & Acquisitions* (*M&A*) with four arguments – acquirer (Hewlett-Packard), acquiree (Palm), money paid ($1.2 billion) and event time stamp (two days ago), and one attribute – event indicator (has been acquired) – a word or a sequence of words that clearly signals about the possible presence of an event.

The EE problem has been the subject of active research for more than twenty years since series of MUC conferences started in 1987. Initially, the EE task was limited so several domains and was characterized by the availability of small-sized corpora. Nowadays, with the rapid evolution of socially-oriented internet, it is becoming a crucial task for businesses to analyze the millions of documents with multilingual content each day with minimal latency in order to get insight and provide critical decisions in time. We believe that modern and effective solution to the EE problem would allow businesses to dramatically minimize the time required to start utilizing new sources of information and reduce the operating costs of such systems.

One of the popular approaches to information extraction (IE) and EE as a sub-problem of IE in particular is in the use of domain specific extraction patterns such as linear extraction patterns for annotation graphs and patterns for dependency-based parse trees (DPT). In this paper we present our approach to semi-automatic generation of linear extraction patterns for annotation graphs while using DPT patterns in the process of constructing the first ones.

The paper is structured as follows. In the next section we briefly outline related work, in particular, we outline papers describing algorithms that minimize human efforts required to build extraction patterns. In Section 3 we describe in details four steps of the algorithm. Section 4 presents the results of experiments. We conclude in Section 5 by summarizing the results and outlining current and future work.

1. Related Work

One of the most known algorithms that learn multi-slot extraction patterns for free texts is Whisk [1]. It requires a corpus of documents annotated with syntactic data.

The problem of learning conventional character-based regular expressions from annotated corpora is described in [2]. The authors demonstrate the applicability of their method in extracting such structured entities as software names, urls, phone and course numbers. It is still an open question if their approach can be effectively applied for extracting higher level structures such as relations or events.

Snowball [3–5] and ExDisco [4, 6] are examples of algorithms for learning information extraction patterns from un-annotated corpora.

Snowball is a logical successor of DIRPE [3–5, 7] which was developed to extract binary relations from semi-structured web data. Snowball as DIRPE extracts binary relations but it is targeted for free texts. The main idea is that there is a small seed of highly reliable known relations which are searched in a corpus. Once these relations are found, the algorithm generalizes sentence context around the found relations and creates patterns. These patterns are used to extract new, previously not seen, relations. New relations are added to the initial seed, and the algorithm iteratively repeats the steps. It stops when no new relations have been found.

ExDisco uses different approach to deal with the problem of absence of annotated corpora. It divides the corpus into two parts (relevant and irrelevant documents) and uses the following idea: relevant documents should contain relevant events and relevant events are contained in the relevant documents. ExDisco uses a small seed of 2 or 3 known patterns to make an initial split of documents, does syntactic analysis of every sentence and uses its results for pattern generation.

As opposed to conventional IE from relatively small annotated corpora, Open Information Extraction (OIE) deals with large Web-sized un-annotated sets of documents. The two most known information extraction systems that do OIE are KnowItAll [4, 8, 9] and TextRunner [4, 8, 10]. They rely on syntactic information rather than lexical data.

2. Generating Linear Patterns

The motivation behind this research is to minimize human efforts in constructing event extraction patterns for new types of events. There are three assumptions underlying our algorithm. The first one is the way how we define the verity of events. We define an event to be true (positive) if its indicator and arguments linguistically represent a valid event mention no matter what polarity, modality, etc. the event mention has. For instance, in the sentence "Reuters announces that HP will not acquire Palm" the pair {Reuters, announces} represents a valid event mention of the type *Company Announcement* while the pair {HP, announces} does not. On the other hand, the triple {HP, Palm, will not acquire} represents a valid event mention of the type *Mergers & Acquisitions* while the triple {Palm, Reuters, will not acquire} does not.

The algorithm requires that documents are annotated with named entities that might be arguments of events. This includes not only such named entities as persons, companies, positions and event indicators, but also temporal and monetary expressions. The complete set of required named entities depends on types of events to be extracted. We use the existing NERs (in particular, we use OpenNLP library) which we trust. It means that we consider texts annotated by appropriate NERs as gold standard and we do not consider confidences of extracted entities (if NERs provide such information) during the process of evaluating the extraction patterns.

We also assume that if two events are extracted by the same DPT pattern they both are either true or false. More generally, a group of N events all extracted by the

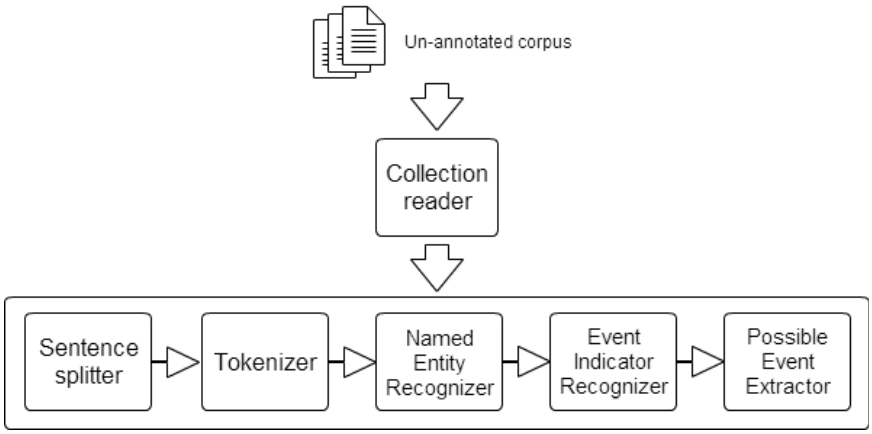

Fig. 1. Processing pipeline

same DPT pattern contains either only true or false events. A verity of a group of N events is determined by a verity of a random sample drawn from it. This is the main assumption that allows us to minimize human efforts required to build extraction patterns by validating only such a small random sample but not the whole entire group that might contain tens of thousands of events.

As we have mentioned, the algorithm is composed of four steps. At the first step, we extract possible events from an un-annotated corpus of documents.

2.1. Extracting Possible Events. Any event type is described by an indicator (word that clearly signals about the event presence, usually, a verb) and arguments together with their semantic types. For instance, an event of type *Mergers & Acquisitions* is described by an event indicator of type *AcquisitionIndicator* and two arguments – *acquirer* of type *Company* and *acquiree* of type *Company*. Another example is an event of type *PersonAnnouncement* that is described by an indicator of type *AnnouncementIndicator* and one argument *announcer* of type *Person*. Formally, an event is defined as the following triple:

$$E\left[T, I, \{A_i^N, A_i^T\}_{i=1}^m\right], \tag{1}$$

where T is the event type, I is the type of event indicator, event has m arguments and i-th argument has name A_i^N and type A_i^T.

The core idea at the first step is to generate all possible events for every sentence in the corpus. To do it, we have built the processing pipeline depicted in Fig. 1. The primary task of this pipeline is to recognize in text the instances of event indicators and named entities that might be arguments of events.

The pipeline has typical architecture intended for extracting named entities. Initially, it splits a text into sentences and tokens. After that, it uses Named Entity and Event Indicator Recognizers to extract named entities and event indicators. We use OpenNLP library to extract named entities and a dictionary-based recognizer to extract event indicators.

The last component on the pipeline (Possible Event Extractor) uses previously extracted information together with descriptions of events to extract possible events. First, this component determines if a sentence may contain at least one event of any type. To do it, it iterates over description of event types. For each event type, it determines if a sentence contains (1) an indicator of appropriate type and if so (2) a minimal number of appropriate named entities. For instance, for event of type *PersonAnnouncement*

a sentence must contain at least one indicator of type *AnnouncementIndicator* and at least one named entity of type *Person*. For event of type *Mergers & Acquisitions*, a sentence must contain at least one indicator of type *AcquisitionIndicator* and at least two named entities of type *Company*. If a sentence cannot contain a mention of at least one event, it is not processed further.

If there can be at least one event mention in a sentence, we apply Stanford parser to obtain DPT. For every type of event T for which there can be at least one event mention, we generate all possible events. At the second step, we construct a list of all appropriate event indicators $\{I\}_{i=1}^{n}$ found in a sentence. Then we construct a set of named entities that will be part of possible events. We compute the shortest paths in DPT from indicators to every named entity in this set. We then generate all possible events around each event indicator. For every possible event, we construct the DPT extraction pattern. We consider its string representation to be possible event's pattern id – a non unique string that is composed of event type and DPT paths. Two events of the same type having the same paths from indicators to arguments will have the same pattern id. The format of pattern id is as follows: $IndicatorType\ Attribute_1Type:Path_1$, $Attribute_2Type:Path_2, \ldots$ For instance, the event mentioned in the sentence "Google acquires Neotonic Software" has the following pattern id: *AcquisitionIndicator Company:nsubj, Company:dobj*.

It is possible to count the number N of events that are generated around each indicator. Let us denote the number of distinct types of arguments as $m^{/}$ and for each j-th distinct type let S_j be the number of arguments in the event definition of this type and let N_j be the number of named entities of this type in a sentence. Then there can be $P_j(N_j, S_j) = N_j!/(N_j - S_j)!$ variants to fill S_j arguments with N_j named entities. To compute the number of possible events, we need to multiply all these values $N = \prod_{j=1}^{m'} P_j(N_j, S_j)$.

2.2. Grouping Possible Events into Clusters. After an input corpus has been processed and all possible events have been identified, we group events according to their pattern ids. Inside every group all events have the same pattern id. In other words, they are all extracted by the same DPT pattern. We sort groups by cardinality and generate random subset for each group.

2.3. User Validation. At the third step, a user validates small random subset of each cluster using Eclipse IDE. Eclipse UIMA CAS Editor plugin [11] visualizes events' annotations and annotations of indicators and arguments. A user is presented with a text annotated with possible events and the task for him is to iterate through every event and mark each event either as true or false. The task thus is to judge if a particular event annotation (possible event) correctly links arguments with event indicator (Fig. 2).

2.4. Generating Linear Patterns. Finally, the system annotates groups of events as positive or negative based on subsets validated by a user at the previous step. This is done by counting the numbers of true and false events in a subset. If there are more positive events in the subset, the whole entire group is annotated as positive and vice versa. Positive groups are further used to generate and generalize event extraction patterns.

An input data for the pattern generation algorithm is the sentence annotation graph. The graph contains such annotations as events, their indicators and arguments (named entities) and sentence context which is presented as tokens' types in the TextMARKER

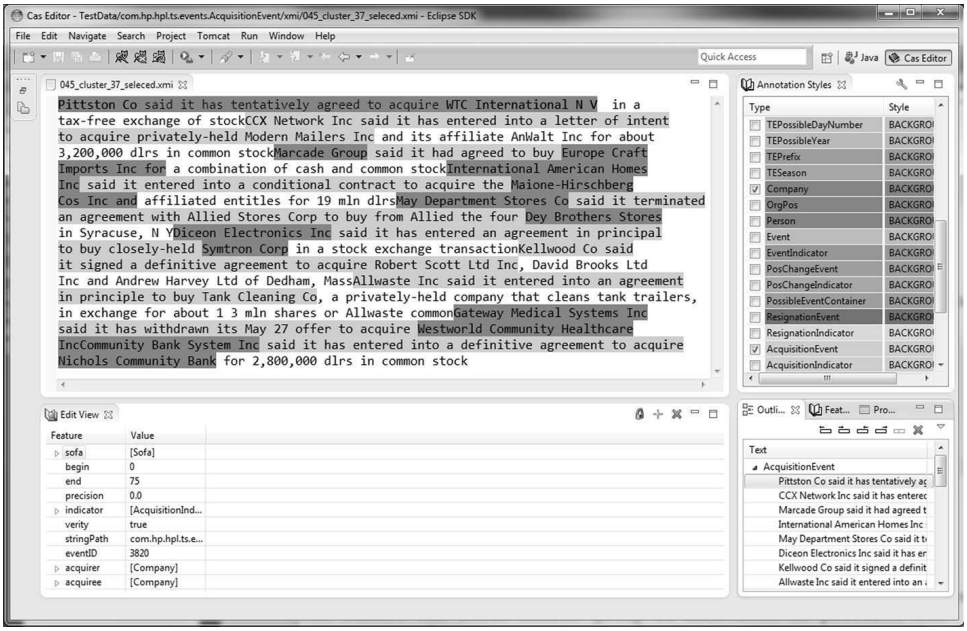

Fig. 2. UIMA CAS Editor perspective (Eclipse IDE)

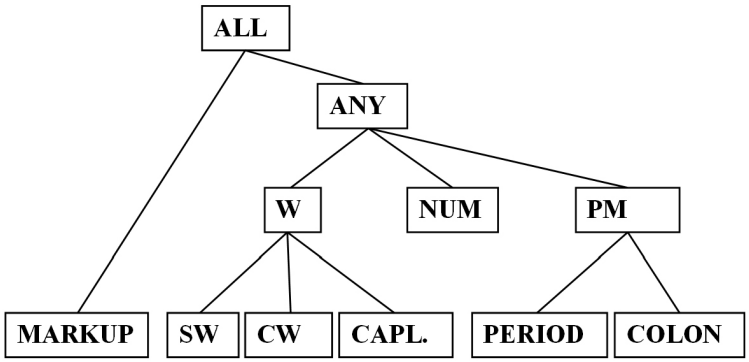

Fig. 3. Type system of TextMARKER rule engine.

typesystem notation (Fig. 3). We use bottom-up generalization strategy and start with the most specific version of a pattern. We generalize patterns until F1-measure of new ones increases.

There are two operations that we use during the pattern generalizing process (Fig. 4):

A. *BottomUpGen*: bottom-up generalization (see Fig. 3). After the token type is changed, its neighbors are checked. If there are two tokens with the same type close to each other, they are replaced by one token of the same type with expanded quantifier: "company" "based" "in" "Tel" "Aviv" → SW SW SW CW CW → SW[3] CW[2] → W[5]

   (a) replacing text of a word with its specific word type: "recently" → SW; "May" → CW; "INC" → CAP

   (b) replacing type of word with a more general type:

```
1: procedure GeneralizeRule(rule)
2:     cur_F1 ← 0
3:     new_F1 ← TestRule(rule)
4:     while (new_F1 − cur_F1) > 0 do
5:         cur_F1 ← new_F1
6:         rules ← empty_list
7:         rules.add(BestRule(BottomUpGen(rule)))
8:         rules.add(BestRule(QuantModif(rule)))
9:         if NewRulesUnknown(rules) then
10:            newRule ← BestRule(rules)
11:            new_F1 ← TestRule(newRule)
12:            rule ← newRule
13:        end if
14:    end while
15:    return (rule)
16: end procedure
```

Fig. 4. Algorithm of pattern generalization

SW, CW, CAP → W

(c) replacing punctuation marks with more general type:
COLON, COMMA, SEMICOLON → PM

(d) replacing types with a more general type:
W, PM → ANY

B. *QuantModif*: quantifier modification (the expansion/restriction of a quantifier)

(a) Expansion: SW[3] → SW[2,4]

(b) Restriction: CW[2,7] → CW[2,6]

Using these two operations we iteratively generalize patterns. At every step we select the best pattern and we stop generalizing process when F1-measure stops improving (new_F1). We keep the list of previously generalized patterns. If the current pattern has already been seen, it is not processed further. This corresponds to line 9 of the algorithm (Fig. 4).

3. Experiments

We ran preliminary experiments on the English part of the Reuters (RCV2) dataset. We have processed this dataset and have determined (a) the sentences that contain possible events and (b) actual number of true events in those sentences. Table 1 presents the properties of constructed in such a way annotated dataset. For instance, we have

Table 1. Properties of annotated dataset constructed based on English part of RCV2

Event type	M&A	MPC	Res
Sentences with possible events #	83	45	80
True events #	12	16	25

determined that there are 83 sentences that may contain events of type *Mergers & Acquisitions* (M&A) and only 12 true events are mentioned in those 83 sentences. The table also presents the results for events of types *Management Position Change* (MPC) and *Resignation* (Res).

Table 2. Train/Test partitions of annotated dataset

	Partition 1	Partition 2
M&A	61/22	55 /28
MPC	38/7	30/15
Res	54/26	56 /24

Table 3. Performance metrics for *Mergers & Acquisitions* event type

		handcrafted (10 patterns) Train / Test	automatic(8 patterns) Train / Test
Partition 1	Precision	0.50 / –	**1.00** / **1.00**
	Recall	0.11 / 0.00	**1.00** / **0.33**
	F1-measure	0.18 / –	**1.00** / **0.50**
Partition 2	Precision	– / 0.50	**1.00** / **1.00**
	Recall	0.00 / **0.50**	**1.00** / **0.50**
	F1-measure	– / 0.50	**1.00** / **0.66**

Table 4. Performance metrics for *Management Position Chang* event type

		handcrafted (3 patterns) Train / Test	automatic(9/8 patterns) Train / Test
Partition 1	Precision	0.78 / 0.50	**0.92** / **1.00**
	Recall	0.58 / **0.33**	**1.00** / **0.33**
	F1-measure	0.67 / 0.40	**0.96** / **0.50**
Partition 2	Precision	0.67 / **1.00**	**1.00** / **1.00**
	Recall	0.60 / **0.40**	**1.00** / **0.40**
	F1-measure	0.63 / **0.57**	**1.00** / **0.57**

In the first experiment, we have compared handcrafted patterns with patterns obtained automatically and provided their performance metrics on both train and test sets. The handcrafted patterns have been constructed previously based on the RSS articles obtained from such sources as Yahoo and Google news feeds. Due to the fact that we have obtained quite small dataset, we have made two train/test partitions of entire annotated dataset in each creating randomly train/test sets in proportion 2:1 (see Table 2).

Tables 3–5 present the results of experiments for each event type separately. For the M&A event type (Table 3), eight automatically constructed patterns outperformed or produced the same results as ten handcrafted ones in terms of precision, recall and F1-measure on both train and test sets in both partitions. Note that in the first partition on test and in the second partition on train data, the handcrafted patterns did not extract any event, that is why we do not provide performance metrics for these cases. The algorithm constructed the same number of patterns (eight) in both partitions.

For the MPC event type (Table 4) we observed the same situation. The automatically constructed patterns had the same or higher performance as the handcrafted ones. However, the algorithm constructed 9 patterns in the first and 8 patterns in the second partitions compared to the three handcrafted patterns. Moreover, in the second partition on test set, the automatically created and handcrafted patterns demonstrated the same performance.

The results for the Res event type are presented in Table 5. In this case, the handcrafted patterns had higher precision by lower recall and F1-measure.

Table 5. Performance metrics for *Resignation* event type

		handcrafted (5 patterns) Train / Test	automatic(10 patterns) Train / Test
Partition 1	Precision	**1.00 / 1.00**	0.90 / 0.33
	Recall	0.19 / 0.25	**0.90 / 0.75**
	F1-measure	0.32 / 0.40	**0.90 / 0.46**
Partition 2	Precision	**1.00 / 1.00**	0.83 / 0.50
	Recall	0.18 / 0.33	**0.86 / 0.67**
	F1-measure	0.30 / 0.50	**0.84 / 0.57**

Table 6. F1-measure values of semi-automatically constructed patterns

M&A	MPC	Res
0.4523	0.5428	0.5000
0.4666	0.5714	0.5988
0.4888	0.6750	0.6321
0.5555	0.6904	0.6426
0.8333	0.7023	0.6764

Table 7. Performance metrics of handcrafted patterns on whole dataset

	M&A	MPC	Res
Precision	0.5	0.73	1.0
Recall	0.08	0.53	0.2
F1-measure	0.14	0.62	0.33

In the second experiment, we tested the handcrafted and automatically constructed patterns on the whole annotated dataset. Since as we have mentioned that the dataset is quite small, we tested the algorithm using 5-fold cross validation 5 times. Performance characteristics are depicted at table 6 that presents F1-measure values sorted in increasing order. As it can be seen we got very different result each time (F1-measure: 0.4523–0.8333 for M&A, 0.5428–0.7023 for MPC, 0.5000–0.6764 for Res).

The results of the handcrafted patterns on the whole dataset are presented in Table 7.

4. Conclusions and Future Work

In the paper we outlined the current progress in developing the algorithm for semi-automatic generation of linear event extraction patterns for free texts and presented preliminary experimental results. Our next steps are to enhance the algorithm with additional capabilities and validate it on a larger dataset.

Currently, we extract only mandatory events' arguments. We will add capability to generalize patterns that will include optional arguments as well (such as temporal and monetary expressions). We plan to explore the possibility of improvement and optimization of the algorithm. We will work on enhancing the generalization algorithm by providing additional operations as well as intelligent selection of operations to apply at each iteration. We will also explore possibility to utilize negative examples. We have a dataset of approximately 110000 news articles that we will use for validating the algorithm. We will also study in much more details the theoretical properties of the proposed algorithm.

Резюме

Д.А. Дзендзик, С.В. Серебряков. Автоматизированное построение линейных правил для извлечения событий из неаннотированного текста.

В статье описывается автоматизированный подход к построению линейных правил для извлечения событий из неаннотированных текстов. Алгоритм состоит из четырех шагов: автоматическое извлечение потенциальных событий из корпуса неаннотированных документов, кластеризация их с использованием путей в дереве зависимостей, проверка случайно выбранных примеров из каждого кластера и построение линейных правил на основе кластеров, получивших положительную оценку. Проводится сравнение полученных правил с системой, использующей правила, построенные экспертом вручную.

Ключевые слова: извлечение событий, линейные правила, регулярные выражения, TextMARKER, RUTA.

References

1. *Soderland S.* Learning Information Extraction Rules for Semi-Structured and Free Text // Machine Learning. – 1999. – V. 34, No 1–3. – P. 233–272.
2. *Li Y., Krishnamurthy R., Raghavan S., Vaithyanathan S., Jagadish H.V.* Regular expression learning for information extraction // EMNLP'08 Proc. Conf. on Empirical Methods in Natural Language Processing. – Stroudsburg, PA, USA: Association for Computational Linguistics, 2008. – P. 21–30.
3. *Agichtein E., Gravano L.* Snowball: extracting relations from large plain-text collections // DL'00 Proc. Fifth ACM Conf. Digital libraries. – N. Y., USA: ACM, 2000. – P. 85–94.
4. *Bach N., Badaskar S.* A Review of Relation Extraction. – 2007. – URL: http://www.cs.cmu.edu/ñbach/papers/A-survey-on-Relation-Extraction.pdf.
5. *McDonald R.* Extracting Relations from Unstructured Text. Technical Report: MS-CIS-05-06. – 2005. – URL: http://www.ryanmcd.com/papers/MS-CIS-05-06.pdf.
6. *Yangarber R., Grishman R., Tapanainen P.* Automatic Acquisition of Domain Knowledge for Information Extraction // COLING'00 Proc. 18th Conf. on Computational linguistics. – Stroudsburg, PA, USA: Association for Computational Linguistics, 2000. – V. 2. – P. 940–946.
7. *Brin S.* Extracting Patterns and Relations from the World Wide Web // WebDB'98 Selected papers from the Int. Workshop on The World Wide Web and Databases. – London, UK: Springer-Verlag, 1999. – P. 172–183.
8. *Etzioni O., Banko M., Soderland S., Weld D.S.* Open information extraction from the web // Communications of the ACM. – 2008. – V. 51, No 12. – P. 68–74.
9. *Etzioni O., Cafarella M., Downey D., Kok S., Popescu A.-M., Shaked T., Soderland S., Weld D.S., Yates A.* Web-scale information extraction in knowitall: (preliminary results) // WWW'04 Proc. 13th Int. Conf. on World Wide Web. – N. Y., USA: ACM, 2004. – P. 100–110.
10. *Yates A., Banko M., Broadhead M., Cafarella M.J., Etzioni O., Soderland S.* TextRunner: Open Information Extraction on the Web // NAACL-Demonstrations'07 Proc. Human Language Technologies: The Annual Conference of the North American Chapter of the Association for Computational Linguistics: Demonstrations. – Stroudsburg, PA, USA: Association for Computational Linguistics, 2007. – P. 25–26.

11. *Kluegl P., Atzmueller M., Puppe F.* Integrating the Rule-Based IE Component TextMarker into UIMA // Proc. LWA. – 2008. – P. 73–77.

Поступила в редакцию
31.07.13

Dzendzik, Darya – Student, Saint-Petersburg State University; Intern, Hewlett-Packard Laboratories, Saint Petersburg, Russia.

Дзендзик Дарья Анатольевна – студент, Санкт-Петербургский государственный университет; практикант, Российское отделение исследовательской лаборатории "Hewlett-Packard Laboratories", г. Санкт-Петербург, Россия.

E-mail: *daryadzen@gmail.com*, *daria.dzendzik@hp.com*

Serebryakov, Sergey – PhD, Research Engineer, Hewlett-Packard Laboratories, Saint Petersburg, Russia.

Серебряков Сергей Валерьевич – кандидат технических наук, инженер-исследователь, Российское отделение исследовательской лаборатории "Hewlett-Packard Laboratories", г. Санкт-Петербург, Россия.

E-mail: *sergey.serebryakov@hp.com*