R2E: Rule-based Event Extractor

Jakub Dutkiewicz, Maciej Nowak, Czeslaw Jedrzejek

Institute of Control and Information Engineering,
Poznan University of Technology,
M. Sklodowskiej-Curie Sqr. 5, 60-965 Poznan, Poland
{firstname.lastname}@put.poznan.pl

Abstract. In this paper we present a rule-based method of event extraction from the natural language. We use the Stanford dependency parser in order to build a relation graph of elements from input text. This structure along with serialized extraction frames is converted into a set of facts. We describe a process of creation of application of rules, which aims to match elements from the text with corresponding slots in the extraction frames. A possible match is derived by the comparison of verbal phrases from the text with lexicalizations of anchors (constituting the most important part of each frame) stored in an ontology. The rest of the extraction frame is filled with other elements of the dependency graph, with regard to their semantic type (determined by lexicalizations of allowed types defined in frames and ontology) and their grammatical properties. We describe conversions required to create a consistent knowledge base of text phrases, classification of semantic types and instantiated slots from the extraction frames. We use the Drools engine in order to extract events from such a knowledge base.

Keywords: event extraction, natural language processing, frames

1 Introduction

Event extraction is the induction of n-ary relations from natural language texts. Some event extraction systems focus on the text coverage – the more events extracted the better. There exist ACE and ERE [1] annotation standards specified by Entities, Relations and Events. Many different strategies for finding event word triggers and slot filling have been applied. Slot filling has been a topic of the Knowledge Base Population (KBP) 2013 task [4] within the Text Analysis Conference workshops that provide the infrastructure for large-scale evaluation of Natural Language Processing technology. All of the mentioned approaches accept events which are semantically nonsensical. The famous Chomsky's problem of "the colorless green ideas which sleep furiously" remains untouched.

Here, we present a specific event extraction methodology for the English language which partially solves this problem. We use sets of acceptable semantic classes for event arguments, which are stored within frames. A frame is defined as a pair (R, S), where R is a semantic type of frame, and S is a set of its slots. Every slot in S is

defined as a pair (Ri, T), where Ri is the relation, and T is a list of allowed semantic types for this slot. Frame languages are fundamental for annotation [1], event extraction [2] and querying [3].

In this paper the semantic types of frames and slots are expressed as facts derived from ontology. The method is an extension of the one dedicated to the Polish language and presented in [5, 6]. Here, the algorithm [6] is transformed into a set of rules dedicated to a stage of processing.

1.1 Note on the Definitions

To simplify notation, we use the number sign (#) to denote entities within a semantic data model; the italic font style to denote predicates; the quotation marks to denote literals; a language of literals is attached after "@" sign in an abbreviated form; *e.g.* #Policeman is a semantic class but "Policeman"@en is the English literal. We use round brackets to express statements, *e.g. #hasLex*(#Policeman, "Policeman"@en). Throughout the paper we use the shorten version of Drools syntax by omitting names of attributes in rules and facts statements. It is due to the space limitations.

2 The System Architecture and its Components

The extraction system consists of four major elements: a taxonomically structured dictionary, a dependency parser, an extraction frames container and a reasoning module equipped with extraction rules. The detailed architecture of the dictionary data is described in section 2.1. We use the Stanford dependency parser [8], a transformation of the parser output into facts is straightforward. An example of this transformation is presented in section 2.2. A specific description of the extraction frames stored in the extraction frame container is presented in section 2.3. A detailed explanation of extraction rules is described in section 2.4. The architecture of the system as implemented in the R2E (Rule-based extraction tool) is illustrated in Fig. 1.

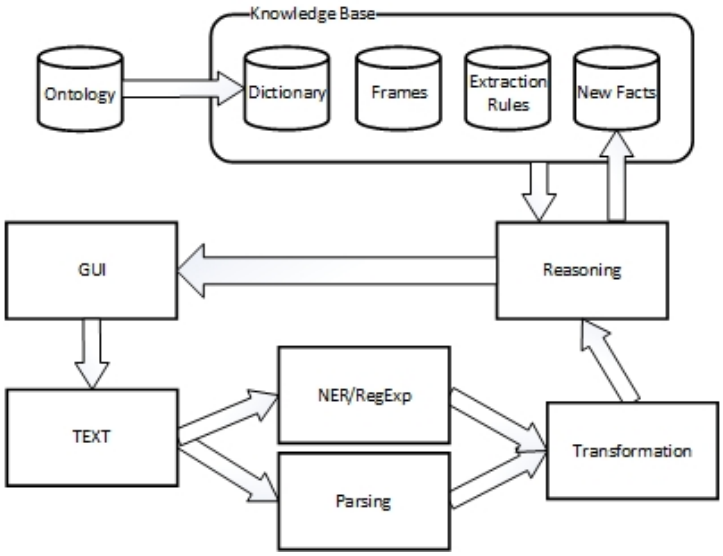

Fig. 1. The R2E tool architecture

2.1 Dictionary Data

Dictionary data are a set of statements used for an event extraction. An initial dictionary is created with use of the T-box of the ontology. Class #Thing is the root of dictionary. #Event, #IsA and #HasLex classes are required as those classes are technical core of the dictionary. The dictionary uses two binary relations – the taxonomical subsumption relation expressed by the *#isA* relation and the lexicalization relation expressed by the *#hasLex* relation. Both a subject and an object of the *#isA* relation are classes. A subject of the *#hasLex* relation is a class, and an object is a literal with a specified language. Together they compose the lexicalization of the class. There are two special technical languages, which have the special meaning –the language @NER and the language @RegExp. The language @NER is used in order to express named entities. Literals in the @RegExp language indicate lexicalizations of classes. The literals in @NER and @RegExp languages are handled by dedicated subprograms.

Table 1 The set of predicates within the basic dictionary.

Subject	Predicate	Object
#Event	*#isA*	#Thing
#Thing	*#hasLex*	"([0-9]|[aA-zZ])*"@RegExp
#Event	*#hasLex*	"event"@en
#IsA	*#isA*	#Event
#IsA	*#hasLex*	"be"@en
#HasLex	*#hasLex*	"lexicalize"@en

The initial dictionary contains a meta-language vocabulary. This vocabulary is sufficient for extending the dictionary. To extract events, the dictionary needs to contain event specific data, which consist of the definition of the event itself as well as a vocabulary for the constraints of the event. Extension of the dictionary for the "Distribution" event is presented in Tab. 2.

Table 2 Extension of the #Distributing event for the basic dictionary.

Subject	Predicate	Object
#Distributing	*#isA*	#Event
#Distributing	*#hasLex*	"distribute"@en
#Distributing	*#hasLex*	"sell"@en
#Inanimate	*#isA*	#Thing
#Product	*#isA*	#Inanimate
#Product	*#hasLex*	"product"@en
#Organization	*#isA*	#Thing
# Organization	*#hasLex*	"ORGANIZATION"@NER

The vocabulary in Tab. 1 and Tab. 2 is sufficient for extracting events of distributing some goods as long as a subject of this event is recognized by the named entity recognition subsystem correctly.

2.2 Transformation of the Stanford Dependency Output into Facts.

The Stanford dependency parser output is specified in [8]. Each dependency is described as a binary relation, where the relation name is equal to type of dependency and attributes of the relation are equal to textual values of tokens within the sentence. Literals which express attributes consist of two parts – the actual literal, which appeared in the text; and an identifier which is equal to the position of this literal in the text. There is an empty line in between sentences.

We transform a dependency output format into contextual facts. Each dependency is transformed into a *dependency* predicate. The actual dependency type and its arguments are stored as arguments of this predicate. A position of the arguments in the sentences stored as a separate argument. On top of that, we store contextual information in the arguments of this predicate. We use two types of contexts. The first one is the technical processing context. The separate context of this type should be used for each message. The second context is the sentence context. Dependencies obtained from one sentence have the same context of this type. The template of the transformed dependency is shown below:

dependency(type, argument1, position1, argument2, position2, processingContext, sentenceContext)

This pattern lets us conveniently design extraction rules and control the activation of certain rules. A transformation process is implemented in Java. Transformed dependencies of a sample message are presented in Tab. 3.

Table 3. A sample transformation of dependencies

Input message	Man kidnapped woman. Terrorist kidnapped prime minister.
Initial context	*processing-context*(“ext”)
Output Dependencies	*nsubj*(kidnapped-2, Man-1) *root*(ROOT-0, kidnapped-2) *dobj*(kidnapped-2, woman-3) *nsubj*(kidnapped-2, Terrorist-1) *root*(ROOT-0, kidnapped-2) *amod*(minister-4, prime-3) *dobj*(kidnapped-2, minister-4)
Transformed dependencies	*dependency*(“nsubj”,”kidnapped”,”2”,”Man”,”1”,”ext”,”1”) *dependency*(“root”,”ROOT”,”0”,”kidnapped”,”2”,”ext”,”1”) *dependency*(“dobj”,”kidnapped”,”2”,”woman”,”3”,”ext”,”1”) *dependency*(“nsubj”,”kidnapped”,”2”,”Terrorist”,”1”,”ext”,”2”) *dependency*(“root”,”ROOT”,”0”,”kidnapped”,”2”,”ext”,”2”) *dependency*(“amob”,”minister”,”4”,”prime”,”3”,”ext”,”2”) *dependency*(“nsubj”,”kidnapped”,”2”,”minister”,”4”,”ext”,”2”)

2.3 Extraction Frames

Let us define extraction data model as a set of facts which is used as a link between a predicative structure of the reasoning data and vocabulary used by the dependency parser output language. The extraction data model consists of subsets called extraction frames. These frames give a detailed specification of the event by defining thematic roles. The specified roles represent topology of an event. These roles are described in a form of slots, consisting of their names, names of the frame, dependency types, lists of allowed semantic types and optional lists of prepositions. The names are only used to distinguish slots from each other. The dependency type indicates one of the predicates used in dependency parser to describe relations between parts of the sentence. The list of allowed semantic types is defined in the corresponding event definition from the dictionary described in section 2.1. A list of prepositions is used to differentiate possible indirect objects. In the example given in Listing 1, we can choose a “weapon” slot by adding “with” preposition.

An extraction frame may be composed of a varying number of slots. However, it must contain the exactly one “anchor” slot. It is a dedicated type of the slot, because it is the first and the necessary condition in the process of extracting events. Only after the anchor slot is found, the rest of the slots are matched with the elements from the dependency graph. In the vast majority of cases, a verbal phrase fulfills the role of an anchor.

Listing 1. Slots within the #Killing frame

```
slot("anchor", #Killing, root, #Killing)
slot("victim", #Killing, dobj, #Human)
slot("perpetrator", #Killing, nsubj, [#Human, #Organization])
slot("weapon", #Killing, pobj, [#Weapon, #PhisicalObject], "with")
```

After the reasoning process, we receive filled instances of the slots. Instances of slots are stored with *slotInstance* predicate, which possess the matched textual value and additional attributes related to a phrase for which the slot instantiation was performed on: id of a word and a context indicator (distinct for each analyzed sentence). The examples of filled slot instances are presented in section 4.

The last part of a system related to extraction frames is a set of lexicalizations. We can divide this set into two kinds of statements: lexicalizations of anchors and lexicalizations of dictionary entries. The first group consists of examples of anchors, specific for a given extraction frame, and is connected to the aforementioned frame. The second group is crucial in the slot matching process in case the NER module does not find appropriate group for the analyzed phrase. Additionally, the system offers the use of regular expressions when defining lexicalizations. Using @RegExp keyword in our dictionary, we can express, that each uppercase phrase followed by the ".Inc" string should be interpreted as an instance of #Organization class. These types of expressions enhance the process of matching phrases from the dependency graph with allowed semantic types of slots. The exemplary lexicalization statements are presented in Listing 2.

Listing 2. Lexicalizations for anchor of the #Killing frame and the #Weapon class

```
#hasLex(#Killing, "killed")
#hasLex(#Killing, "murdered")
#hasLex(#Weapon, "rifle")
#hasLex(#Weapon, "gun")
```

3 Extraction Rules

Let us define extraction rules as a set of rules, which convert the parsed sentences into event specific data. The extraction rules use data provided by dependency parser, extraction frames data and dictionary data. The extraction rules are specific for given structures of parsing data. As a result the extraction rules determine how instances of slots are asserted into the knowledge base. The template for a slot instance is shown below.

```
slotInstance(EventClass, TypeOfSlot, textualValue, entryID, extractionContext, sentenceContext, rootID)
```

There are two main types of rules: the event recognition rules are designed to recognize the appearance of the event in text; the slot matching rules are designed to supplement recognized events with specific slots. The basic event recognition rule is presented in Listing 3. The first condition in the illustrated rule accepts only roots of the sentence, which are supposed to be verbs*. The second condition accepts every slot, which is an anchor. The third part of the conditions is accepted only if the semantic type of the event is lexicalized by the verb chosen in the first condition. The lLast condition – extraction mode, is specified in section 3.1. This rule is activated only for slots with the "anchor" literal as a type of slot.

Listing 3. Basic event recognition rule

```
if(
      dependency("root","ROOT", "0", ?Lit, ?n, ?ExCtx, ?SCtx)
      slot(?Name, ?SemanticType, "anchor", "ROOT", "")
      hasLex(?SemanticType, ?Lit))
      extraction-mode("basic")
then
      insert(
            slotInstance(?Name, "anchor", ?Lit, ?n, ?ExCtx, ?SCtx, ?n,
      "basic"))
```

Listing 4 illustrates the basic slot matching rule. This rule is very similar to the Listing 3 rule with one exception – Rule 2 is activated only if the *slotInstance* of the root has already been asserted. ID if the root is stored within the arguments of the output assertion.

Listing 4. Basic slot matching rule.

```
if(
      slotInstance(?eventName, "anchor", ?lit1, ?id, ?exCtx, ?sCtx,
?rootID)
      dependency(?dep, ?lit1, ?id, ?lit2, ?id2, ?exCtx, ?sCtx)
      slot(?slotName, ?eventName, ?dep, ?semClass, "")
      hasLex(?semClass, ?lit2)
      extraction-mode("basic")
)
then(
      insert(
            slotInstance(?eventName, ?slotName, ?lit2, ?id2, ?exCtx,
      ?sCtx, ?rootID, ?mode)))
```

Besides the rules in Listing 3 and Listing 4, there are extraction rules which handle sophisticated sentences, passive voice or complex definitions of slots, such as:

- Sentences with more than one verb phrase.
    - Verb phrases are connected with a conjunction
    - Complex sentences
- Passive nominal subjects
- Passive auxiliary verbs
- Prepositional phrases as slots

Complexity of those rules is similar to the ones presented in Listing 3 and Listing 4. At least one rule for each of mentioned grammatical properties is required to perform a proper extraction within one extraction mode (a total of 7 rules). For specific sentence structures, such as phrasal verbs or events with constraints on the slots, additional rules must be provided. Currently we are using 22 rules to perform extraction within one extraction mode. The modal auxiliary verb modifiers and negation are handled on a different level of the process, which is described in section 3.1.

3.1 Extraction Modes

The extraction modes are specified with a special predicate called *extraction-mode*. Every rule that is meant to be activated in a certain mode has this predicate included within its conditions. List of available modes is specified below.

- The basic mode – This is the default mode. It provides the extractions as it is specified in the previous section. In this mode instances of the slots are always single words. Word modifiers are stored within the dependencies. This mode is expressed as *extraction-mode(*"basic").

- The Join mode – In this mode instances of the slots are filled with entire phrases. The identifier of the phrases is created with the use of the identifiers of words within the phrase. This mode is expressed as *extraction-mode(*"join").

- The Aggressive mode – In this mode, the semantic class of slots, except for the root slot, is not specified as one of the rule conditions. In this mode, the extractor generates greater number of less accurate extractions. This mode is expressed as *extraction-mode*("aggro").

- The Dictionary mode – In this mode, the extractions of *#IsA* and *#HasLex* events feed the dictionary data. As the extractions from complex sentences are not always correct and the philosophical meaning of the copula verbs is ambiguous, it is advised to use this mode with special caution. This mode is expressed as *extraction-mode*("dict").

- The Modality and negation mode – In this mode, the verbs modified by auxiliary modal verbs or modified by negation are not extracted. It allows us omitting the supposed events or the explicitly negated events. This mode is expressed as *extraction-mode*(“modneg”).

Multiple modes are not allowed, a combination of modes is expressed within a single statement. *E.g.* extraction-mode(“aggro-join”) is the correct mode – in this mode, the slot instances are filled with entire phrases, and the dictionary is not checked for a non-root slots. Rules for this mode are separate from the rules for “aggro” and “join” modes.

4 Examples

For the purpose of evaluation of our method, we decided to present the whole extraction process using two exemplary sentences, containing event descriptions. In this section we describe and present the conversion from Stanford output format (dependency graph of elements in the sentences) into facts written according to Drools syntax [7]. We enrich the knowledge base with statements derived from our dictionary and the extraction frames serialized into Slots. As the final result of the reasoning process, we present filled instances of Extraction Frames.

4.1 Sample Extraction in “basic” Mode

In this example, extractor was set to the *extraction-mode*(“basic”). Stanford dependency parser is shown in the Listing 5; newly created facts are visible in the Listing 6. The example input sentence is shown below.

Foreign terrorists killed minister of Bolivia with the sniper rifle.

Listing 5. The Stanford parser output

```
amod(terrorists-2, Foreign-1)
nsubj(killed-3, terrorists-2)
root(ROOT-0, killed-3)
dobj(killed-3, minister-4)
prep(minister-4, of-5)
pobj(of-5, Bolivia-6)
prep(killed-3, with-7)
det(rifle-10, the-8)
nn(rifle-10, sniper-9)
pobj(with-7, rifle-10)
```

Listing 6. The extracted facts.

```
slotInstance(#Killing, "anchor", "killed", "3", "ext", "1", "3", "basic")
slotInstance(#Killing, "perpetrator", " terrorists ", "2", "ext", "1",
"3", "basic")
dependency("amod", "terrorists", "2", "Foreign", "1", "ext", "1")
slotInstance(#Killing, "victim", "minister", "4", "ext", "1", "3",
"basic")
dependency("prep", "minister", "4", "of", "5", "ext", "1")
dependency("pobj", "of", "5", "Bolivia", "6", "ext", "1")
slotInstance(#Killing, "weapon", "the sniper rifle", "8-10", "ext", "1",
"3", "basic")
dependency("det", "rifle", "10", "the", "8", "ext", "1")
dependency("nn", "rifle", "10", "sniper", "9", "ext", "1")
```

4.2 Sample Extraction in "join" Mode

The extractor mode is set to *extraction-mode*("join") in this example. The Stanford dependency parser output is shown in the Listing 7; these are newly created facts. The newly created facts are visible in the Listing 6. The input sentence is shown below.

GoodsForYou Inc. sells and delivers chemical products for exclusive members.

Listing 7. The Stanford parser output

```
nn(inc.-2, GoodsForYou-1)
nsubj(sells-3,Inc.-2)
root(ROOT-0, sells-3)
cc(sells-3, and-4)
conj(sells-3, delivers-5)
nn(products-7, chemical-6)
dobj(sells-3, products-7)
prep(products-7, for-8)
amod(members-10, exclusive-9)
pobj(for-8, members-10)
```

Listing 8. The extracted facts.

```
slotInstance(#Distributing, "anchor", "delivers", "5", "ext-join", "1",
"join")
slotInstance(#Distributing, "supplier", "GoodsForYou Inc.", "1-2", "ext-
join", "1", "join")
```

```
slotInstance(#Distributing, "recipient", "exclusive members", "9-10",
"ext-join", "1", "join")
slotInstance(#Distributing, "object", "chemical products", "6-7", "ext-
join", "1", "join")
slotInstance(#Selling, "anchor", "sells", "3", "ext-join", "1", "join")
slotInstance(#Selling, "seller", "GoodsForYou Inc.", "1-2", "ext-join",
"1", "join")
slotInstance(#Selling, "buyer", "exclusive members", "9-10", "ext-join",
"1", "join")
slotInstance(#Selling, "object", "chemical products", "6-7", "ext-join",
"1", "join")
```

4.3 Quantitative evaluation

To test the method, we equipped the system with data sufficient for extraction of 16 separate events - *#Injuring; #Stealing; #Punishing; #Accessing; #Killing; #Kidnapping; #Purchasing; #Recommending; #Hiding; #Delivering; #Poisoning; #Searching; #Attacking; #Breaking In; #Meeting; #Travelling, #Rescheduling*. We have used taxonomies and vocabulary, which were applied in our previous work [5, 6]. We manually annotated 100 MUC [13] messages (each consisting from 3 to 30 sentences) with the mentioned events. To diminish the influence of low quality vocabulary stored within the dictionary, we have used the aggressive mode extraction. Within the 100 messages we marked 162 events. The system was capable of extracting 113 events correctly and 33 events incorrectly. However, within the 113 events only 55 had all the semantic slots instances matched correctly. 16 events had one semantic slot instance error, 19 events had two slot errors, and 23 events had more than 2 slot errors. Measures for the evaluation are presented in Tab. 4.

Table 4 Evaluation measures for the quantitative evaluation for 16 events

	Precision	Recall	F-measure
Event recognition	77%	70%	73%
Event extraction (no errors)	38%	34%	35%
Event extraction (up to 1 error)	49%	44%	46%
Event extraction (up to 2 errors)	62%	56%	58%

The most significant factor that has influenced the extraction is the quality of vocabulary stored within taxonomies. We are currently working on extending the dictionary with external resources.

5 Conclusions

In this paper we presented a modified version of our event extraction system. There are numerous advantages coming from the introduction of rule-base extraction over the algorithmic extraction used in our previous work [5, 6]. First of all, this type of methodology is much easier to configure. A modification of the existing rules and an addition of new extraction rules is far less complicated than remodeling of the algorithm. Another advantage is the use of a knowledge base as the data storage. Because of that, in the future releases of our system we plan to incorporate query answering as a form of exploration of the knowledge base. On top of that we plan to develop a natural language interface for the system configuration and provide more comprehensive vocabulary and taxonomical data. The last positive aspect of rule-based approach is the homogeneity of data. In our solution parsed input text, dictionaries derived from the ontology, extraction frames and rules are all expressed in the same way (the Drools syntax [7]), in order to enable the reasoning. Our work is similar to several other methods, in particular these applied in Boxer [9], FRED [10] and Lemon [11]. We use the RDF triple format for the specification of the dictionary; as it is done in FRED. However our system does not accept events with nonsensical classes of attributes. Equipping our knowledge base with Wordnet data would provide sufficient dictionary for the extraction. Incorporating lexical data from external sources is elaborately specified in Lemon. Our tool has the most important features listed for extractor functionalities on page 4 in [12], namely no 10. semantic role labeling, no 11. event detection and no 12. frame detection.

The web-link to the project site is https://github.com/PUTR2E/R2E.

Acknowledgement. This work was supported by the Polish National Centre for Research and Development (NCBR) No O ROB 0025 01 and 04/45/DSPB/0105 grants.

References

1. Aguilar J., Beller C., McNamee P., Van Durme B., Strassel S., Song Z. and Ellis J.: A Comparison of the Events and Relations Across ACE, ERE, TAC-KBP, and FrameNet Annotation Standards. ACL Workshop: EVENTS (2014)
2. Hobbs, J. and Riloff, E.: "Information Extraction", Handbook of Natural Language Processing, 2nd Edition (2010)
3. Lally A., Prager J. M., McCord M. C., Boguraev B., Patwardhan S., Fan J., Fodor P., and Chu-Carroll J.: Question analysis: How Watson reads a clue. IBM Journal of Research and Development 56(3): 2 (2012)
4. Proceedings of the Sixth Text Analysis Conference (TAC 2013), NIST, Gaithersburg, Maryland, USA; http://www.nist.gov/tac/publications/2013/papers.html (2013)

5. Dutkiewicz J. and Jedrzejek C.: Ontology-based event extraction for the Polish Language, Proceedinds of 6th LTC'13 Eds. Vetulani Z., Uszkoreit H., Fundacja AMU, pp. 489–493 (2013)
6. Dutkiewicz J., Falkowski M., Nowak M., and Jedrzejek C.: Semantic Extraction with Use of Frames, PolTAL 2014 – 9th International Conference on Natural Language Processing will take place September, Warsaw, Poland, in print. (2014)
7. Drools rule engine, drools.jboss.org
8. Marie-catherine De Marneffe, C. D.: Stanford typed dependencies manual. (2008)
9. Curran J.R., Clark S., and Bos J.: Linguistically Motivated Large-Scale NLP with C&C and Boxer. Proceedings of the ACL 2007 Demonstrations Session (ACL-07 demo), pp. 33–36 (2007)
10. Draicchio F., Gangemi A., Presutti V., Nuzzolese A. G.: FRED: From Natural Language Text to RDF and OWL in One Click. ESWC (Satellite Events): 263–267 (2013)
11. McCrae J., Spohr D., and Cimiano P.: Linking lexical resources and ontologies on the semantic web with lemon. In Proceedings of the 8th extended semantic web conference on The semantic web: research and applications, Berlin, Heidelberg, pp. 245–259 (2011)
12. Gangemi A.: A Comparison of Knowledge Extraction Tools for the Semantic Web. ESWC: 351–366 (2013)
13. Fourth Message Understanding Conference, (MUC-4): Proceedings of a Conference Held in McLean, Virginia (1992)