SOFTWARE—PRACTICE AND EXPERIENCE
Softw. Pract. Exper. 0000; **00**:1–32
Published online in Wiley InterScience (www.interscience.wiley.com). DOI: 10.1002/spe

Ontology Population for Open-Source Intelligence: a GATE-based Solution

Giulio Ganino[1], Domenico Lembo[1], Massimo Mecella[1,2], Federico Scafoglieri[1]

[1] *Dipartimento di Ingegneria Informatica, Automatica e Gestionale, Sapienza Università di Roma, Italy*
`[lastname]@diag.uniroma1.it`

[2] *Laboratorio Nazionale di Cyber Security, CINI, Italy*
`massimo.mecella@consorzio-cini.it`

SUMMARY

Open-Source INTelligence (OSINT) is intelligence based on publicly available sources such as news sites, blogs, forums, etc. The Web is the primary source of information, but once data are crawled, they need to be interpreted and structured. Ontologies may play a crucial role in this process, but, due to the vast amount of documents available, automatic mechanisms for their population are needed, starting from the crawled text. This paper presents an approach for the automatic population of predefined ontologies with data extracted from text, and discusses the design and realization of a pipeline based on the General Architecture for Text Engineering (GATE) system, which is interesting for both researchers and practitioners in the field. Some experimental results that are encouraging in terms of extracted correct instances of the ontology are also reported. Furthermore, the paper also describes an alternative approach and provides additional experiments for one of the phases of our pipeline, which requires the use of predefined dictionaries for relevant entities. Through such a variant the manual workload required in this phase was reduced, still obtaining promising results. Copyright © 0000 John Wiley & Sons, Ltd.

Received . . .

KEY WORDS: General Architecture for Text Engineering (GATE), Ontology Population, Information Extraction, Internet as a data source

1. INTRODUCTION

Open-Source INTelligence (OSINT) is intelligence, i.e., information gathering, based on publicly available sources such as news sites, blogs, forums, etc. OSINT is nowadays used in many application scenarios, for instance fo security (e.g., identifying lone wolves and weak signals on the Web [1]), market intelligence (understanding users' profiles and trends), or statistics (to cross-check and complement data collected with traditional methods [2]). A major issue in using *Internet as a data source* is that Web data come mainly in the form of free text, thus with no structure and formal semantics. This means that two problems have to be faced. First, how to select structured information from unstructured texts, and second how to interpret the selected information according to precise semantics.

For a comprehensive solution we have investigated how to populate a domain ontology using the information extracted from a given cluster of textual documents crawled from the Web. Ontologies are, indeed, nowadays recognized as the best way to represent domain knowledge at a conceptual level, and thus are particularly suited to define concepts and relationships of interest for a given application [3]. Structuring Web data according to the predicates and axioms defined in an ontology turns out to be particularly effective for our purposes, even in the light of the reasoning abilities ontologies allow for [4]. We would like to point out that the task of extracting (intensional/schema level of) ontologies from documents in an automatic way was not here taken into consideration. Domain ontologies usually have a very complex intentional structure to satisfy all the needs of the specific application domain at hand, and typically their design cannot be completely automatized [3, 5]. In this work we, instead, have pursued an approach for information extraction that assumed that a domain ontology was already available, and that text should be mined to extract instances of ontology predicates.

In order to solve this task, *Named Entity Recognition* (NER) [6] can be initially adopted. However, this approach only focuses on recognizing specific types of concepts (e.g., persons, places, or organizations), without considering relationships among them. The *Relational Information Extraction* attempts, in addition, to identify the relationships that exist among concepts, as allowed by, e.g., SystemT [*], an IBM-developed commercial tool that is used in order to derive rich information from documents and emails. Our approach, thus, adopted both NER and Relational Extraction techniques, using open-source technologies.

Among various existing open-source tools for information extraction (e.g., LingPipe[†] or OpenNLP[‡]), we decided to use GATE[§], given the flexibility it allows to customize its underlying architecture, and to incorporate other external components developed by third parties. The extraction activity in GATE is typically carried out through different stages, each depending on the contingent needs of the user, who can, for instance, adopt existing dictionaries (a.k.a. Gazetteers) for NER, or create new ones, and /or specify tailored extraction rule sets through the use of the *Java Annotation Pattern Language* (JAPE) [7]. GATE has become popular in the last years, especially in relation to information extraction from English documents. To some extent, it also supports other languages, primarily thanks to the dictionaries created and then shared on the platform by its many users.

The main contributions of our work are *(i)* to show how to build a complete ontology population pipeline step-by-step, presenting all the relevant methods, techniques, and design choices, entirely based on open source tools; *(ii)* to provide an experimental validation of our approach that shows its performances in terms of the quality of the information we are able to extract; *(iii)* to describe how to exploit semantic technologies to reduce the manual workload needed in some components of our pipeline (in particular, for the definition of Gazetteer lists). The use of such semantic technologies opens new research challenges to be possibly addressed in future works. Our contributions are addressed primarily to practitioners who are interested in constructing a GATE-based pipeline for information extraction, and, in particular for the ontology population. At the same time we believe that our work, and in particular contributions *(ii)* and *(iii)*, provides some advancements in information extraction and, thus, it is also addressed to researchers in this field.

Our techniques have been tested within the XASMOS and RoMA projects, involving the 'Leonardo' company (formerly Selex), and the 'Sapienza' Research Center on Cyber Intelligence and Information Security. The projects focused on OSINT for security applications. Namely, we considered the case study 'Mafia Capitale', from the name of an important 2015 investigation that received quite a lot of attention from the Italian media, and, thus, turned out to be a valid testbed (for both number of Web documents available and significance of the domain). For our case study specific dictionaries and JAPE rules for Italian were created to instantiate a domain ontology with the information extracted from Web documents. We present our case study to provide some insight on the possibilities offered by such an approach. In particular, we successfully applied it to more than 2600 documents crawled from the Web. The results we obtained were encouraging in terms of number of extracted instances and quality of the information extracted.

In our case study we experienced that some of the tasks we had to deal with, such as dictionaries or JAPE rule definitions, were rather domain specific and time-consuming since they required a

[*]http://researcher.watson.ibm.com/researcher/view_group_subpage.php?id=5577
[†]http://alias-i.com/lingpipe/
[‡]https://opennlp.apache.org/
[§]https://gate.ac.uk/

lot of manual work. We, thus, started to investigate how to refine our approach so that the time needed to perform these tasks could be reduced and the solution adopted could be easily reused in different contexts. In particular, we focused on the dictionary construction task, and faced it with a more general approach, which relies on a simple extraction of dictionaries through SPARQL queries issued over the open knowledge base Wikidata [*].

The paper is structured as follows: Section 2 reports some background; Section 3 describes our approach for ontology population and introduces the modules that were used in the GATE system; Section 4 presents our case study and explains the specific actions we had to take for the application at hand; Section 5 reports the evaluations and results for our case study. Section 6 illustrates the Wikidata-based approach for the generation of Gazetteer lists, whereas Section 7 discusses the evaluations and results for this approach. Section 8 reports our conclusions.

2. BACKGROUND

When considering documents as a data source we may resort to the techniques developed in the fields of Information Retrieval (IR), Information Extraction (IE) and, more recently, to Question Answering (QA) and Text Understanding (TU) [8]. This section recalls some basic notions and tools that will be used in the following and that are at the basis of our approach.

2.1. Natural Language Processing (NLP)

NLP aims at analyzing, identifying, and solving problems related to the automatic generation and understanding of human language. Through computational linguistic techniques, it addresses the most peculiar challenges related to the ambiguity of the language (since the same statement may have more than one meaning), its flexibility (given by the different ways in which a fact can be described), and the stream of updates (given by the continuous creation of new words) [9]. The process carried out by NLP to decode and understand unstructured information is made up of several critical steps:

- *Tokenization.* This task breaks up a character string into words, punctuation marks and other meaningful expressions. This process starts with breaking down the raw text in tokens, which can be words, spaces, sentences or dots [10]. After this stage, words are not classified into grammatical categories, and there is a very limited indication of the syntactical structure of the text, but there is still a reasonable amount of meta-information that can be added to the documents [11].
- *Part of Speech (POS).* This is a further stage in text analysis to associate every token with a grammatical category or POS. This phase aims at labeling each word with a unique tag which indicates its syntactical role, i.e. Noun, Verb, or Pronoun.
- *Named Entity Recognition (NER).* NER labels atomic elements in the sentence into categories (such as "Person" or "Location") through the application of specific rules and statistical machine-learning techniques [6]. In turn, NER follows two steps:
  1. *Segmentation*, i.e., the identification of the bounds of the analyzed entities (i.e., start and end offset);
  2. *Classification*, i.e., the assignment of the analyzed entities to a specific category.

  In order to identify and classify the entities, some approaches can be taken into consideration [12]:

  – *Lookup lists and Gazetteers*. These approaches resort to pre-computed lists of entities divided into categories;

[*]`https://www.wikidata.org/`

  - *Rule-based (pattern-matching).* The entities are pinpointed through rules that analyze the context and/or the key characteristics of the entities, such as orthography, POS classification, etc;
  - *Machine-trainable.* Machine learning techniques are used in this approach, such as those based on Hidden Markov Models (HMM).

- *Semantic Role Labelling (SRL).* SRL assigns a semantic role to a syntactical constituent of a sentence, and adds further annotations to words in the documents with respect to those identified in the previous steps. In other terms, the aim of SRL is to understand the meaning of an entire sentence starting from the meaning of each individual word and the relationship that exists among words [13].

2.2. General Architecture for Text Engineering (GATE)

GATE is an architecture, a framework and a development environment for Language Engineering (LE) [14, 15]. Since it is an 'architecture', it defines the organization of an LE system and the assignment of responsibilities to different components, and ensures that the interactions of the components satisfy the system's requirements. As a 'framework', it provides a reusable design for an LE software system and a set of preset software building blocks that language engineers can use, extend and customize for their specific needs. As a 'development environment', it helps its users to minimize the time they spend building new LE systems or modifying existing ones, by supporting overall development and providing a debugging mechanism for new modules [14, 15].

GATE has a component-based model which allows for easy coupling and decoupling of the processors, thereby facilitating the comparison of alternative configurations of the system or different implementations of the same module (e.g., different parsers). GATE comprises a core library and a set of reusable LE modules. The framework implements the architecture and provides (amongst other things) facilities for processing and visualizing sources, including representation, import and export of data. The provided reusable modules are able to perform basic language processing tasks such as POS and semantic tagging. This eliminates the need for users to keep recreating the same kind of components, and provides a good starting point for new applications.

GATE components may be implemented through a variety of programming languages, but they are always represented to the system as Java classes. A a class may simply call the underlying program or provide an access layer to a database; alternatively it may implement the whole component.

The modules that were used in our work are described in more detail in Section 3.

2.3. Ontologies

In Computer Science, an ontology is commonly defined as a specification of a conceptualization, that is, a formal description of an abstract, simplified view of a certain portion or aspect of the world [16, 17]. According to this definition, an ontology provides a conceptual representation of a domain of interest, and thus it abstracts from aspects typical of logical or physical data modeling and storage. Furthermore, an ontology is formal, which means that the description of the world it provides is not ambiguous, since it is usually given in some mathematically based language with a precise syntax and clear semantics, commonly rooted in some kind of logic. Another important characteristic of ontologies is that they are shared, i.e., they are agreed upon by all their users. Therefore, ontologies are considered a good way to represent knowledge on the Web where they are mainly used to add semantics to data. This also allows for the usage of powerful reasoning mechanisms that ontologies are usually equipped with [4]. The importance of ontologies to interpret and structure Web data is also demonstrated by the huge standardization effort carried out by the W3C, which led to the definition of OWL, the standard Web Ontology Language [*].

As it is typical in ontologies and data modeling, in OWL we distinguish between *intensional* and *extensional* knowledge. Intensional knowledge is given in terms of logical axioms involving

[*]`https://www.w3.org/TR/owl2-primer/`

classes (a.k.a. concepts) and properties, which are of two types, *object properties* (a.k.a. binary relationships or roles) and *data properties* (a.k.a. attributes). Classes denote sets of objects, object properties denote binary relations between objects, whereas data properties denote binary relations between objects and values from predefined datatypes. `Person` or `City` are examples of classes, `livesIn` is an example of object property, whereas `personAge` or `cityName` are examples of data properties. Logical axioms can be constructed over a given alphabet of classes, properties, and individuals, i.e., constants denoting objects. As a simple example, one can specify that every individual that lives in a city must be a person, which in formal terms corresponds to the *typing* of the domain of the object property `livesIn`, and that in OWL can be expressed as `ObjectPropertyDomain(livesIn Person)`. At the extensional level, an OWL ontology is a set of (membership) assertions about instances of the ontology. For example, `ClassAssertion(Person John)` indicates that the individual `John` is an instance of `Person`, whereas `ObjectPropertyAssertion(livesIn John NY)` specifies that the pair of individuals `<John,NY>` is an instance of `livesIn`, with the intended meaning that John lives in New York. Moreover, it is also possible to assert that two individuals in fact denote the same object (notice that OWL does not adopt the unique name assumption, i.e., does not impose that different individuals denote different objects). This is done through the use of the `owl:sameAs` predicate. For instance, the assertion `owl:sameAs(NY NewYork)` states that the two individuals `NY` and `NewYork` have to be interpreted as the same object.

To conclude this section on ontologies, we'd like to mention that, besides standard textual syntaxes used to specify ontologies (e.g., the functional style syntax [*] that was used in previous examples), various attempts have been made to devise graphical representations of OWL ontologies, to better understand ontologies and their specifications for those who are not experts of logic or formal languages. Among them, one which is called GRAPHOL [†] is a recent diagrammatic language that is completely graphical, i.e., it does not require to complement diagrams with logical formulas, and that is equivalent to OWL 2, the current version of the OWL standard [18, 19]. Thanks to this equivalence, GRAPHOL completely preserves the OWL ontology specification, but at the same time allows for a more intuitive ontology comprehension. This is the reason why in the following section ontologies will be presented in this graphic formalism rather than through textual syntax.

3. APPROACH AND ARCHITECTURE

By leveraging GATE, a pipeline was built for the extraction of information from text documents. As already mentioned in Section 1, we assume to have a reference ontology that has been designed by analysts/domain experts as input and that needs to be populated with instances of concepts and properties. To achieve this goal through the components in GATE, our approach proceeds through two main phases (cf. Figure 1).

1. *Semantic annotation.* In this phase, annotations are created, i.e., metadata that indicate properties of the text contained in the analyzed documents. At the end of this phase, the annotations will allow us to identify in the text those entities that are indeed instances of the classes and properties of the ontology given as input to our pipeline.

2. *Ontology population.* In this phase, instances of concepts and relationships of the ontology are extracted, starting from the output of the previous phase.

In the rest of this section, details of all the Processing Resources (PRs) used to create our pipeline in GATE are reported.

[*] `https://www.w3.org/TR/owl2-syntax/`
[†] `http://www.diag.uniroma1.it/~graphol/`

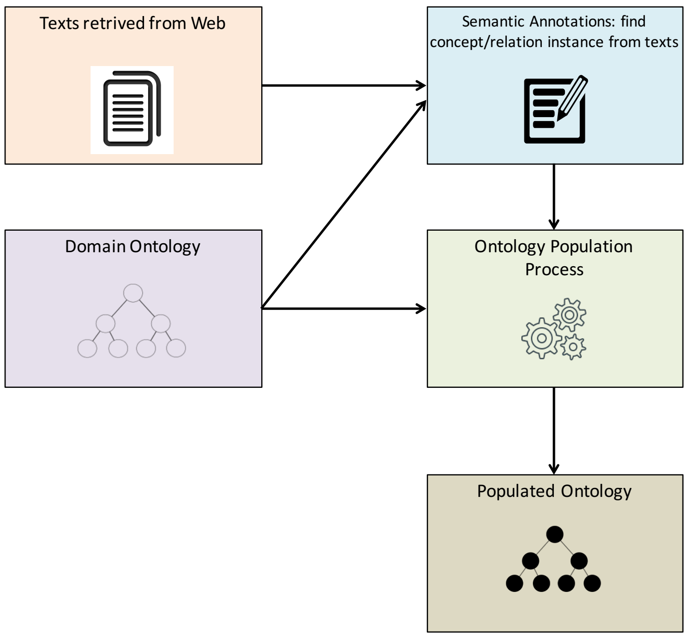

Texts retrived from Web
Semantic Annotations: find concept/relation instance from texts
Domain Ontology
Ontology Population Process
Populated Ontology

Figure 1. The process for ontology population.

3.1. Semantic annotation

Entity identification, entity disambiguation, and text annotation are the three main tasks that the semantic annotation has to take in account. In the GATE-based approach that we propose, this phase relies on several PRs which are available as GATE plugins, some of them provided by third-party organizations[*].

The following pages describe such PRs through an ongoing example in which a short text is analyzed, and show the output produced by each component through some screenshots. In such screenshots, there are three main areas (cf. Figure 2): the top left of the window shows the text that is being analyzed, the right-hand side of the window reports the types of annotations available, and which the user can select, whereas at the bottom there is a description of the selected annotations. This area indicates the *Type* of the annotation, the portion of the text that the annotation refers to (*Start* and *End* characters), the *Id* of the annotation (which is unique), and the *Features* of the annotation (which depend on the type) [†].

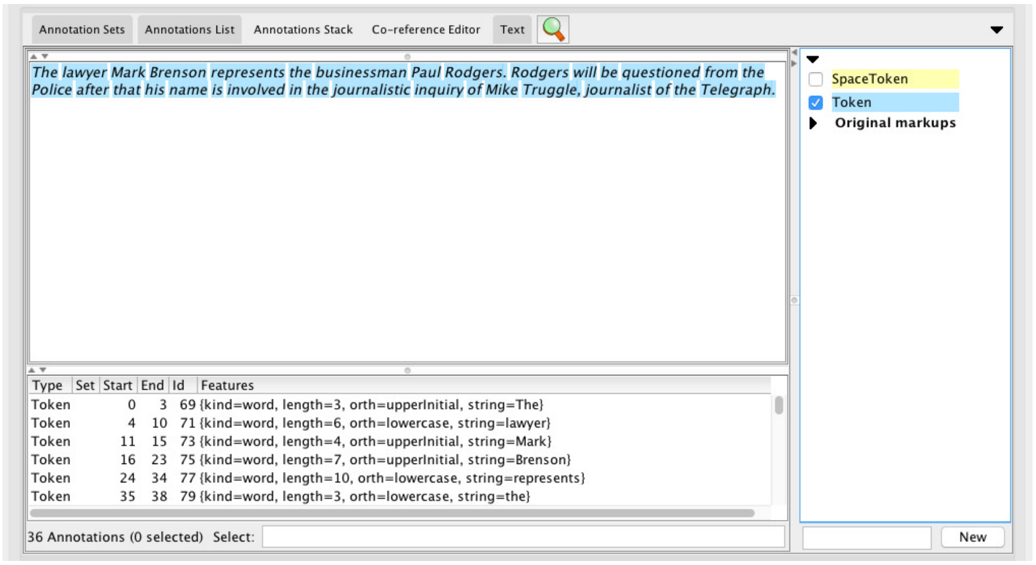
The semantic annotation components used in our approach are the following:

1. *Document Reset*. This component allows to reset the annotations that have been added to a document, and it is useful when different applications are made on the same corpus. This resource has been inserted in the pipeline before any other PRs so that annotations added by a previous application on the text documents do not influence the results obtained in the current execution [7].

2. *GATE Unicode Tokenizer*. This component is used to split the text in *Tokens* and *SpaceTokens*; the latter denote spaces among single terms, whereas the former ones are of four kinds, i.e., number, punctuation, symbol, and word. The use of this component is essential to introduce in the document those annotations that will be exploited in subsequent phases by JAPE rules,

[*] `https://gate.ac.uk/gate/doc/plugins.html`
[†] The column ‘Set’ refers to specific annotation sets, but is not used in this example [7].

which are described later. In Figure 2, we show the output of the GATE Unicode Tokeniser for our ongoing example. For instance, the first row at the bottom describes an annotation of Token type that refers to a string starting at character 0 and ending at character 3. Among the features we can read that the token is of kind word, contains 3 characters, the first character is an upper case letter, and the string is “The”. The same occurs in the other rows.



The lawyer Mark Brenson represents the businessman Paul Rodgers. Rodgers will be questioned from the Police after that his name is involved in the journalistic inquiry of Mike Truggle, journalist of the Telegraph.

Type	Set	Start	End	Id	Features
Token		0	3	69	{kind=word, length=3, orth=upperInitial, string=The}
Token		4	10	71	{kind=word, length=6, orth=lowercase, string=lawyer}
Token		11	15	73	{kind=word, length=4, orth=upperInitial, string=Mark}
Token		16	23	75	{kind=word, length=7, orth=upperInitial, string=Brenson}
Token		24	34	77	{kind=word, length=10, orth=lowercase, string=represents}
Token		35	38	79	{kind=word, length=3, orth=lowercase, string=the}

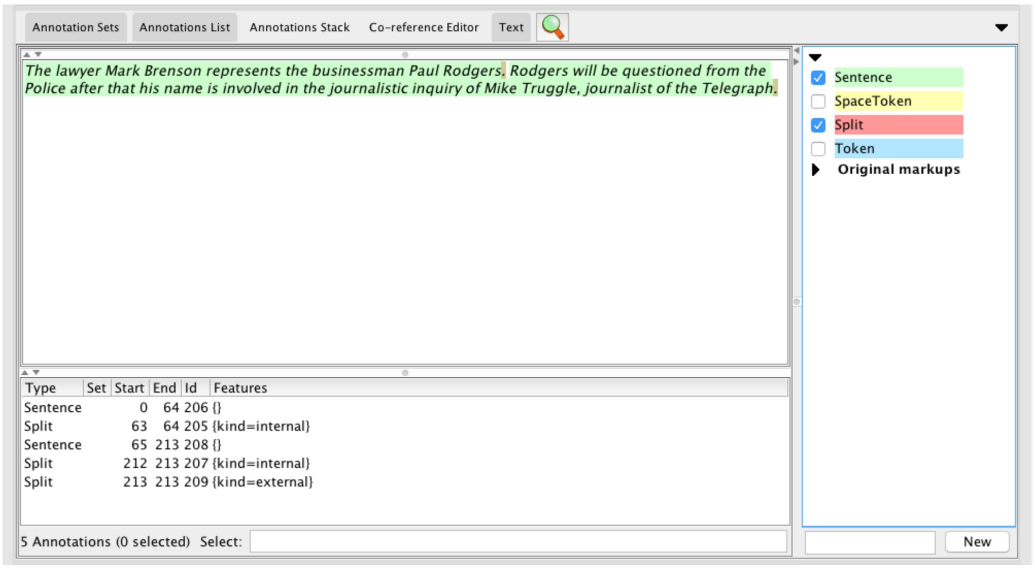
36 Annotations (0 selected) Select:

Figure 2. A GATE Unicode Tokenizer example output.

3. *RegEx Sentence Splitter*. This component divides the processed document into *sentences*, which are chunks of text that make sense in isolation. It is essentially language-independent, in the sense that it can be used, as it is, in many common languages, such as English, German, Italian, etc. It is an alternative component to the ANNIE[*] Sentence Splitter, a splitter component, completely based on JAPE, originally provided by GATE. In particular, RegEx Sentence Splitter outperforms ANNIE in terms of execution time and for its ability to deal with several languages and irregular inputs. Moreover, this component is completely implemented in Java, and, as the name itself suggests, it is based on regular expressions that define the syntactic rules for sentence identification. The RegEx Sentence Splitter adds to the input document two kinds of annotations, i.e., *Sentence*s and *Split*s. As shown in Figure 3, no particular features are assigned to sentence annotations, whereas split annotations can be *(i)* internal, i.e., splits among two sentences, *(ii)* external, i.e., the split that closes the document, or *(iii)* non-splits (not shown in the example), which are fragments similar to splits, but not really splits (such as punctuations used for abbreviations).

4. *TreeTagger Part-of-Speech (TreeTagger POS)*[†]. It is a component for document annotations with POS and lemma information, developed at the Institute for Computational Linguistics of the University of Stuttgart. It is a Markov Model tagger that makes use of a decision tree to obtain more reliable estimates for contextual parameters [20]. It can be used with various languages, provided that it is fed with an input parameter file specific for the language. Despite the fact that there are several POS taggers that could be used in this phase, the TreeTagger POS turned out to be the best one for Italian (which is the language considered in our case study). Indeed, we tested other taggers (such as TagPro [‡]) on various

[*]https://gate.ac.uk/sale/tao/splitch6.html#chap:annie
[†]http://www.cis.uni-muenchen.de/~schmid/tools/TreeTagger
[‡]http://hlt-services2.fbk.eu/textpro/?p=89



The lawyer Mark Brenson represents the businessman Paul Rodgers. Rodgers will be questioned from the Police after that his name is involved in the journalistic inquiry of Mike Truggle, journalist of the Telegraph.

```
Type      Set Start End  Id  Features
Sentence        0    64 206 {}
Split          63    64 205 {kind=internal}
Sentence       65   213 208 {}
Split         212   213 207 {kind=internal}
Split         213   213 209 {kind=external}
```

5 Annotations (0 selected) Select:

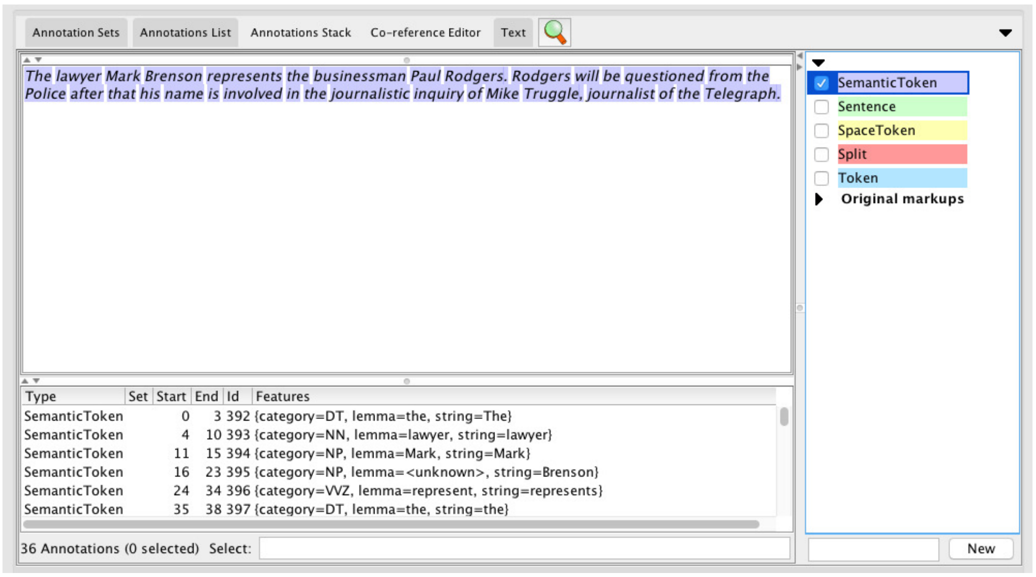
Figure 3. A RegEx Sentence Splitter example output.

documents in Italian, and TreeTagger POS provided the best performance for both the number of correct POS annotations and lemma information. In particular, information on lemmas, which are canonical forms of sets of words (e.g., “represent” is the lemma of “represents”, “representing”, etc.), is a peculiar characteristic of TreeTagger POS. Similarly to the tokenization obtained thorough the GATE Unicode Tokenizer, at the end of this phase there is a set of annotations associated to each token. In this case, the column *Features* provides POS and lemma information. This is shown in Figure 4, where such annotations are called *SemanticToken*s, and where each annotation reports, in particular, a category and a lemma. We notice that the category assumes one among several POS values, specific for the language at hand (e.g., for English, DT is a determiner, NN is a common name, NP is a proper singular name, VVZ, is a verb in the present tense, 3rd person singular).

5. *Gazetteer* [7]. This component annotates the documents on the basis of a set of lists containing names of entities, such as countries or organisations, but also names or abbreviations for types of companies (e.g., Ltd., Corp., Inc.), for political offices (e.g., President, Prime Minister, Senator), etc. Each list can be associated with so-called major and minor types. Normally, at least the major type is specified. Intuitively, these types correspond to categories (or classes of the ontology), and minor types are more specific than the corresponding major types. If the document contains a string matching with an element of a Gazetteer list, the component annotates the string with the major and minor type of the list. If the string has more than one match, the major and minor types of all the matching lists are added.

   As an example, let us consider two different lists called day.list and month.list, respectively. The former contains the days of the week, the latter contains the months of the year. We associated the value *time* as major type to both lists, whereas we set as minor type *days* for the day.list and *month* for the month.list. If a document contains the string ‘Monday’, it will be annotated with a *Lookup* annotation having major type *time* and minor type *day*. This allows to exploit such annotations in following phases (in particular through JAPE rules) at different levels of abstraction, e.g., considering the string as a generic time entity or as a day, depending whether the major or minor type is accessed.

   The effectiveness of this phase is completely dependent on the quality of the information contained in the lists. The construction of such lists is a demanding task, but once a list is created it can be used in several applications. Figure 5 shows the output of the Gazetteer



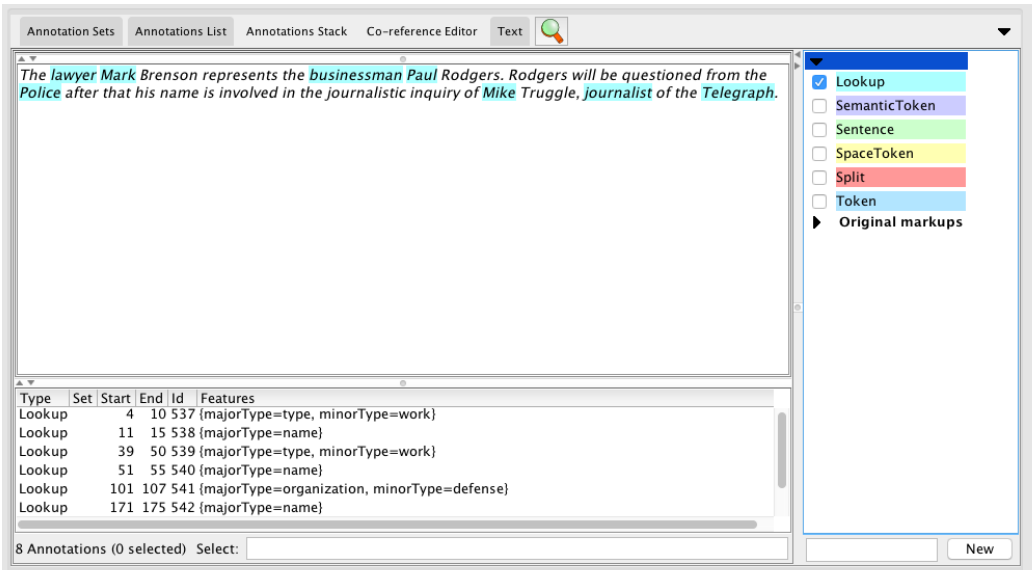
The lawyer Mark Brenson represents the businessman Paul Rodgers. Rodgers will be questioned from the Police after that his name is involved in the journalistic inquiry of Mike Truggle, journalist of the Telegraph.

Type	Set	Start	End	Id	Features
SemanticToken		0	3	392	{category=DT, lemma=the, string=The}
SemanticToken		4	10	393	{category=NN, lemma=lawyer, string=lawyer}
SemanticToken		11	15	394	{category=NP, lemma=Mark, string=Mark}
SemanticToken		16	23	395	{category=NP, lemma=<unknown>, string=Brenson}
SemanticToken		24	34	396	{category=VVZ, lemma=represent, string=represents}
SemanticToken		35	38	397	{category=DT, lemma=the, string=the}

36 Annotations (0 selected) Select:

Figure 4. A TreeTagger POS example output.

PR applied to the text of our ongoing example. We used a list with some job names, a list with some person's proper names, and a list with some names of organizations. For instance, in the fifth row, the string 'Police' is annotated with minor type *defence*, and major type *organization*.



The lawyer Mark Brenson represents the businessman Paul Rodgers. Rodgers will be questioned from the Police after that his name is involved in the journalistic inquiry of Mike Truggle, journalist of the Telegraph.

Type	Set	Start	End	Id	Features
Lookup		4	10	537	{majorType=type, minorType=work}
Lookup		11	15	538	{majorType=name}
Lookup		39	50	539	{majorType=type, minorType=work}
Lookup		51	55	540	{majorType=name}
Lookup		101	107	541	{majorType=organization, minorType=defense}
Lookup		171	175	542	{majorType=name}

8 Annotations (0 selected) Select:

Figure 5. An example of output of the Gazetteer PR.

6. *JAPE Transducer (Semantic)*. This component completes the semantic annotation phase and is used to import the user-written JAPE rules into the GATE platform, so that they can be used to create or modify text annotations. The JAPE language allows to recognize regular expressions among the annotations previously produced. Once the expression is matched, a further annotation referring to the searched patterns/entities is added to the document [7].

The JAPE Transducer executes a JAPE grammar, which is a set of *phases* organized in a precise order, each one made up of a set of patterns and action rules. These phases are performed in a sequential way to create a cascade of finite-state transducers. Each of these transducers takes the output of the previous phase as input, together with the related annotations. The running order of the phases is defined in an indexed JAPE file.

Let’s consider the following example of a phase in JAPE:

1 *Phase: University*
2 *Input: Lookup Token*
3 *Options: control = appelt*
4 *Rule: University1*
5 *Priority:80*
6 *(Token.string == ”University” Token.string == ”of” Lookup.minorType == city):orgName*
7 $\rightarrow$
8 *:orgName.Organization = {rule = “University1”, kind = “university”}*

From line 1 to line 3 there is a header that contains the attributes:

- *Phase*. It indicates the name of the phase.
- *Input*. It indicates the kind of data obtained from previous operations on which the user can perform actions. In the example, we consider *Lookup* annotations created by the Gazetteer.
- *Options*. It indicate the kind of matching style, which defines how we deal with annotations that overlap, or where multiple matches are possible for a particular sequence. There are 5 options:
  - brill: if one or more rules identify a match in the same portion of the document, they are performed all together. More annotations on the same piece of text can be, thus, identified. All the rules are executed starting from the same position and the next matching will start from the position in which the longest match ends;
  - all: it is similar to brill, but the rule continues the search of a match from the next offset with respect to the currently one found;
  - first: when a rule finds a match it is activated regardless of a possible longest match;
  - once: when a rule has been activated, the entire phase ends after the first match;
  - appelt: only one rule can be activated in the same part of the text, according to precise rules of priority:
    - among all the rules that have a match with the identical initial position, only the one that corresponds to the longest match will be activate;
    - if one or more rules have a match on the same portion of the document, the one with highest priority will be activate;
    - if there is more than one rule with the same priority, the rule that was defined earlier will be activate.

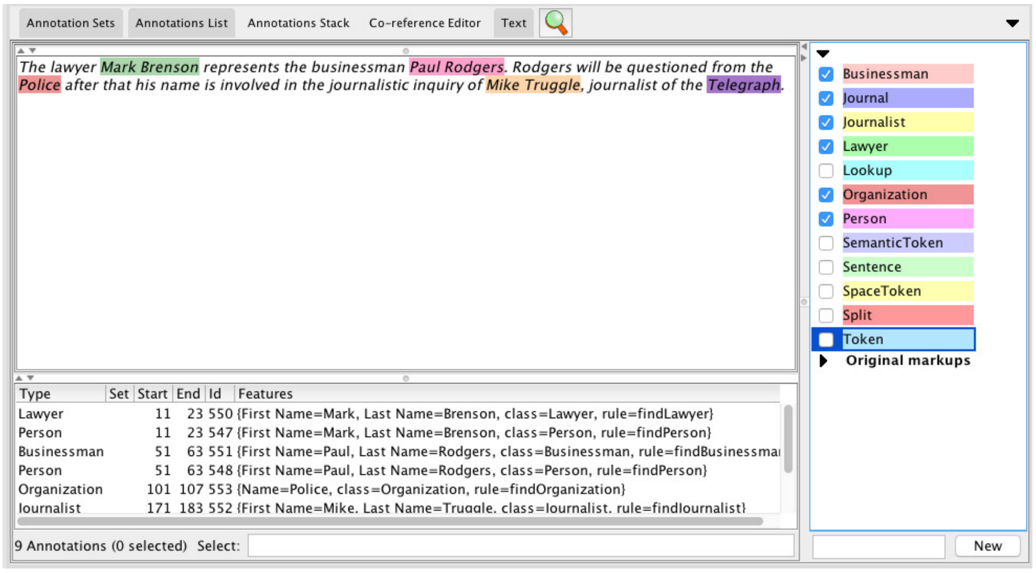
From line 4 to the end there is the description of the actions to take when a matching occurs.

- *Rule*. The fourth line indicates the name of the first rule. The presence of more than one rule in a single phase is allowed but it is advisable to pay attention to the order in which rules are written and performed to prevent unexpected results caused by control options.
- *Priority (optional)*. The fifth line indicates the priority of a rule. The user can declare an optional parameter of priority associated to each rule that is usually a positive integer. A higher number corresponds to a higher priority. If the priority is not declared by default its value is -1.
- *Left Hand Side (LHS)*. The sixth line contains the LHS of the rule, i.e., the code before the arrow. The LHS specifies the pattern that the rule uses to find the match. The definition of a pattern is in brackets. In the example, the pattern is composed by two

consecutive strings 'University' and 'of' followed by a token whose lookup minor type annotation is city. *orgName* is the name associated to the pattern.

- *Right Hand Side (RHS)*. Line 8 reports the RHS, which describes the action to be performed when a match with the pattern defined by the LHS is found in the text. In the example, the entire text fragment involved in the match specified by the LHS (e.g., "University of Rome", if 'Rome' is annotated with minor type city) is annotated as *Organization*. This new annotation has two attributes, namely *rule*, with value *University1* (to indicate the rule producing it) and *kind*, with value *university*. We'd like to point out that the RHS of the rule can even contain Java code to create or manipulate annotations.

At the end of this phase, through the use of specific JAPE rules, the document is usually enriched with new annotations, such as those reported in Figure 6.



The lawyer Mark Brenson represents the businessman Paul Rodgers. Rodgers will be questioned from the Police after that his name is involved in the journalistic inquiry of Mike Truggle, journalist of the Telegraph.

```
Type          Set Start End Id  Features
Lawyer             11   23 550 {First Name=Mark, Last Name=Brenson, class=Lawyer, rule=findLawyer}
Person             11   23 547 {First Name=Mark, Last Name=Brenson, class=Person, rule=findPerson}
Businessman        51   63 551 {First Name=Paul, Last Name=Rodgers, class=Businessman, rule=findBusinessma
Person             51   63 548 {First Name=Paul, Last Name=Rodgers, class=Person, rule=findPerson}
Organization      101  107 553 {Name=Police, class=Organization, rule=findOrganization}
Journalist        171  183 552 {First Name=Mike, Last Name=Truggle, class=Journalist, rule=findJournalist}
9 Annotations (0 selected)  Select:
```

Figure 6. An example output of our JAPE Transducer (Semantic) rules.

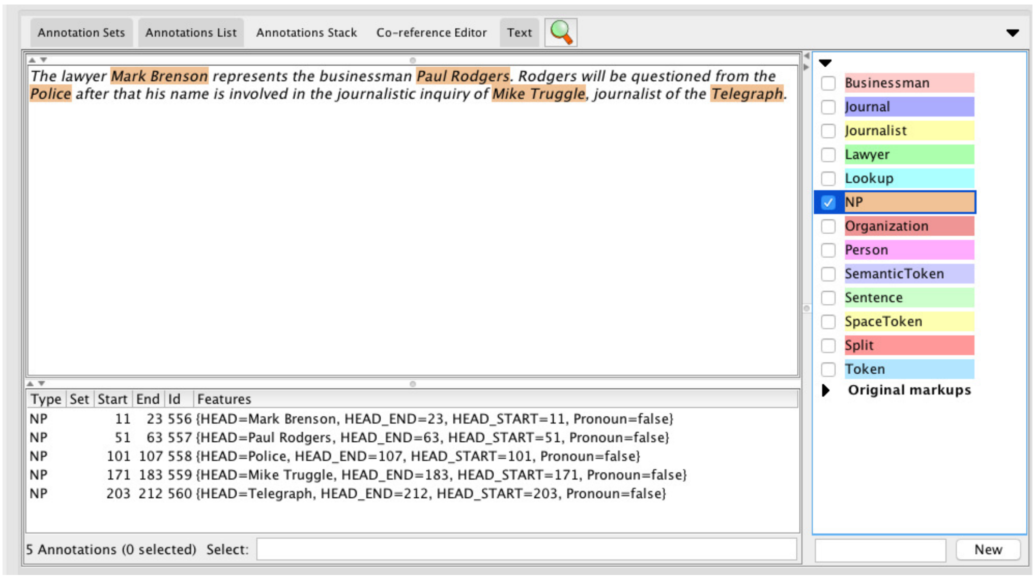
3.2. *Ontology population*

The ontology population is the process of inserting concept and relation instances into an existing ontology. This process does not change the structure of the ontology, i.e., its alphabet and axioms are not modified. What changes are the instances of the concepts and the properties of the ontology. The ontology population requires a reference ontology to be populated and an instance extraction engine, as the latter is responsible for locating in the annotated text instances of concepts and relations [21]. In our approach, the main PR used for the ontology population is the *OwlExporter*[*]. Such PR can populate OWL ontologies by using Protegé-OWL libraries. More precisely, given a document processed by the semantic annotation phase, the OwlExporter exports individuals, data properties and object properties, i.e., it produces class, object property and data property assertions to be added to the OWL file of the ontology. Furthermore, it exports coreference chains, i.e., `owl:sameAs` assertions indicating when different individuals denote in fact the same object of the world.

In addition to the OwlExporter, the ontology population phase in our pipeline uses two other components, and thus it consists of three cascaded PRs, precisely described below.

[*]`http://www.semanticsoftware.info/owlexporter`

1. *JAPE Transducer (MuNPEx)* [*]. This component is implemented in JAPE, and it is used for (multi-lingual) noun phrase (NP) extractions, i.e., identification of elements in a sentence with a noun as head word, which is the word determining the syntactic function of the phrase. It is needed to manage the natural language processing specific for the OwlExporter, as later clarified in the case study. MuNPEx requires a POS tagger to work and can additionally use detected named entities to improve chunking performance [22]. Currently the supported languages are English, German, and French, with additional Spanish support in beta. Thus it had to be adapted for the Italian. For each detected NP, an annotation *NP* was added to the document, as reported in Figure 7 for our ongoing example.



The lawyer Mark Brenson represents the businessman Paul Rodgers. Rodgers will be questioned from the Police after that his name is involved in the journalistic inquiry of Mike Truggle, journalist of the Telegraph.

```
Type Set Start End  Id  Features
NP           11  23 556 {HEAD=Mark Brenson, HEAD_END=23, HEAD_START=11, Pronoun=false}
NP           51  63 557 {HEAD=Paul Rodgers, HEAD_END=63, HEAD_START=51, Pronoun=false}
NP          101 107 558 {HEAD=Police, HEAD_END=107, HEAD_START=101, Pronoun=false}
NP          171 183 559 {HEAD=Mike Truggle, HEAD_END=183, HEAD_START=171, Pronoun=false}
NP          203 212 560 {HEAD=Telegraph, HEAD_END=212, HEAD_START=203, Pronoun=false}
```

5 Annotations (0 selected) Select:

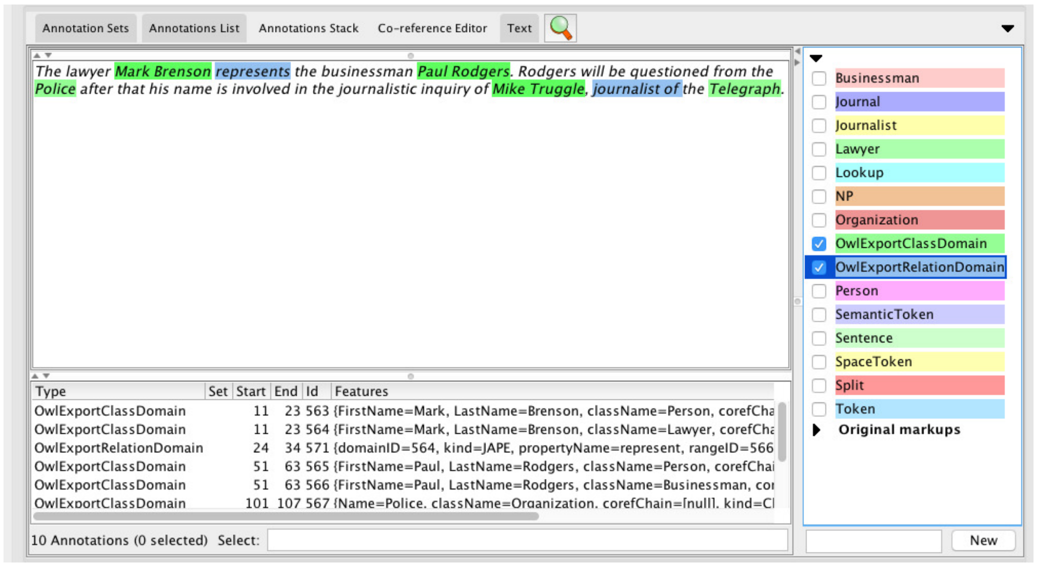
Figure 7. A JAPE Transducer (MuNPEx) example output.

2. *JAPE Transducer (Mapping)*. The annotations created by the semantic annotation phase are not in a compatible format to be directly an input to the OwlExporter. They must be converted through JAPE rules. To this aim, the JAPE Transducer (Mapping) creates the following two new kinds of annotations [23] (as shown in Figure 8):

   - *OwlExportClassDomain*. Annotations of this kind specify which document annotations refers to individuals to be added to the ontology, and which is the class of the ontology they are instance of;
   - *OwlExportRelationDomain*. Similarly to OwlExportClassDomain annotations, annotations of this kind regards the export of instances of object properties, and specify which document annotations identify pairs of individuals to be added as instances of a certain object property of the ontology.

3. *OwlExporter*. It produces OWL membership assertions to be added to the ontology from the document annotations created by the previous PRs. The OwlExporter manages two ontologies. The former is a domain specific ontology that models concepts and relationships that are relevant for a given domain (i.e., it is the ontology we aim to populate). The latter is a domain-independent NLP ontology that contains concepts commonly used in language engineering [24]. By using these two ontologies the OwlExporter is able to link domain-specific entities detected in a text to their lexical representation, e.g., paragraphs, sentences,

[*] `http://www.semanticsoftware.info/munpex`



The lawyer Mark Brenson represents the businessman Paul Rodgers. Rodgers will be questioned from the Police after that his name is involved in the journalistic inquiry of Mike Truggle, journalist of the Telegraph.

Type	Set	Start	End	Id	Features
OwlExportClassDomain		11	23	563	{FirstName=Mark, LastName=Brenson, className=Person, corefCha
OwlExportClassDomain		11	23	564	{FirstName=Mark, LastName=Brenson, className=Lawyer, corefCha
OwlExportRelationDomain		24	34	571	{domainID=564, kind=JAPE, propertyName=represent, rangeID=566
OwlExportClassDomain		51	63	565	{FirstName=Paul, LastName=Rodgers, className=Person, corefChai
OwlExportClassDomain		51	63	566	{FirstName=Paul, LastName=Rodgers, className=Businessman, cor
OwlExportClassDomain		101	107	567	{Name=Police, className=Organization, corefChain=[null], kind=Cl

10 Annotations (0 selected) Select:

Figure 8. A JAPE Transducer (Mapping) example output.

or noun phrases. When the OwlExporter terminates, the population of the reference ontology with the information from the analyzed text documents is obtained.

4. CASE STUDY: DESIGN AND DEVELOPMENT

In this section our case study is described. The focus is, in particular, on the design and development phases. Our solution has been tested in a real project in which we considered the domain of 'Mafia Capitale' from the name of a judicial investigation that involved the City Council of Rome in 2015: alleged criminal organizations misappropriated money allotted to community city services. The investigation received a lot of attention from the media, thus allowing us to collect a great number of newspaper articles. The retrieved information was however completely unstructured and difficult to analyse in an automatic way. At the same time processing it manually was really unapproachable.

We started from the Web crawling phase, from the definition of an ontology to represent some concepts of interest related to our domain, and we created specific Gazetteer lists and JAPE rules used for document annotations by GATE PRs described in Section 3. The technical details on all the phases of the case study, that are very useful for practitioners, and demonstrate the reproducibility of the whole case study, are online at `https://tinyurl.com/Ontology-population-via-GATE`. There is an online appendix with all configuration instructions, the Gazetteer, the JAPE rules, the ontology and the crawled documents, i.e., all the artifacts needed to re-create the case study and the experimental results.

This initial pipeline is referred to in the following sections as OS/HK – open-source/human-knowledge, to mean that the required artifacts have been manually produced, based on human knowledge and also possibly exploiting publicly (open) available information sources (papers, blogs, etc.), especially when creating the Gazetteer lists. Another pipeline in which Gazetteer lists are semi-automatically built by using Wikidata, will be later explored, cf. Section 6.

We would like to underline that the documents that were extracted from the Web were all written in Italian. Thus we could not resort either to the many English-tailored Gazetteer lists available in literature nor to the JAPE rules commonly used for English. We had to define new Gazetteer lists instead, and sets of JAPE rules specific for the present project. At the same time, however, these outcomes are reusable resources, that is, they are generic enough to be exploited even in other

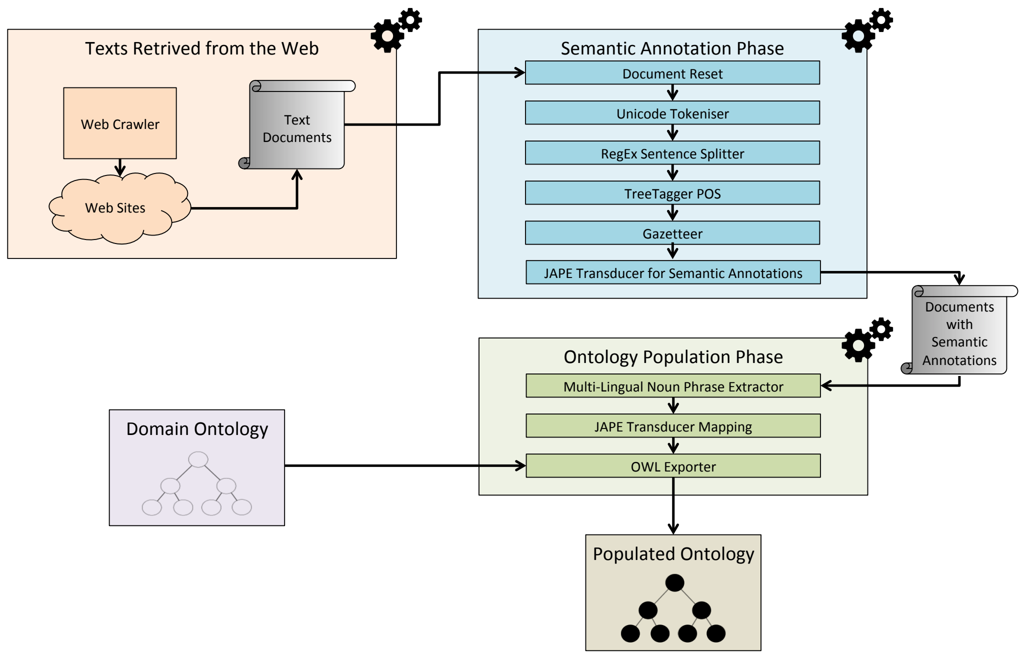

Texts Retrived from the Web
Web Crawler
Text Documents
Web Sites
Semantic Annotation Phase
Document Reset
Unicode Tokeniser
RegEx Sentence Splitter
TreeTagger POS
Gazetteer
JAPE Transducer for Semantic Annotations
Documents with Semantic Annotations
Ontology Population Phase
Multi-Lingual Noun Phrase Extractor
JAPE Transducer Mapping
OWL Exporter
Domain Ontology
Populated Ontology

Figure 9. The description of our Pipeline.

domains where the text to be annotated is in Italian. The interested reader and practitioner can find them in the online appendix mentioned above.

Here follows an overview of the main phases of our project.

4.1. Crawling Phase

Initially, articles from online newspapers were chosen from different Web sources as raw data. Crawling operations were carried out on newspapers for about 8 months, from 16 June 2015 to 29 February 2016. We adopted the Web Content Extractor [*], a software that is able to crawl web sites also in the presence of `robot.txt` files; we accurately configured it through various settings, like the initial seed, which is the URL from which the crawling action starts, and the depth of research. Through these two parameters the crawling software was able to create a SQL table with the following columns:

- *ID* the number of the entry;
- *TITLE* the title text of the article;
- *ARTICLE* the content of the article;
- *URL* the URL where to find the article.

Once this table was populated, it was possible to export the extracted text through SQL queries in XML format, which is the format accepted by GATE.

To obtain articles related to 'Mafia Capitale', for every newspaper Web site that was selected, we set the initial seed to the result of the keyword search we had executed over the newspaper Web site using the words 'Mafia Capitale'. For example, for the Web site of the Italian newspaper called 'La Repubblica', we set the initial seed to "$http://ricerca.repubblica.it/ricerca/repubblica?query = Mafia + Capitale$". It took about 3 hours for the crawling phase to generate 2657 articles, distributed as indicated in the following Table I:

[*] `http://www.newprosoft.com/web-content-extractor.htm`

Newspaper	Number of articles
La Repubblica	938
Il Messaggero	777
Libero	842
Il Fatto Quotidiano	100

Table I. Number of documents in the case study

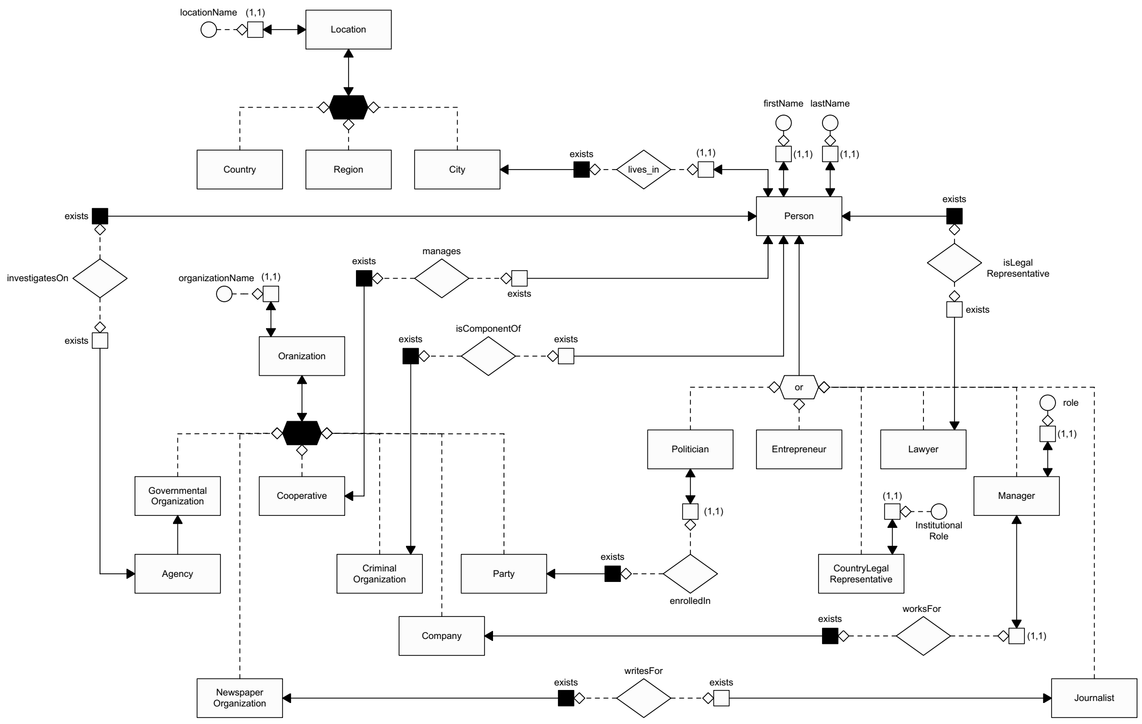


Figure 10. The reference ontology in the case study

4.2. *Domain Ontology*

As anticipated in Section 2, the (intensional level of the) ontology is reported in Figure 10 through the GRAPHOL ontology language, which provides a visual representation of OWL ontologies.

Classes are represented as rectangles, object properties by diamonds, and data properties by circles (as in Entity-Relationship diagrams). To denote the first (domain) and the second (range) component of a property, a white and a black square were respectively used, connected to the properties they refer to with dashed arrows ending with a small diamond. A black hexagon indicates a disjunctive union of classes, whereas the hexagon labeled with 'or' denotes a union (not necessarily disjunct). In both cases, each class in the union is connected to these symbols with a dashed edge. Labels associated to the domain and range of properties are used to indicate cardinality restrictions (similar to cardinality constraints in Entity-Relationship). Solid arrows denote inclusions between classes, both atomic, which correspond to the labeled rectangles, and complex, which are obtained by the use of operators like the hexagons and the squares described above.

The ontology conveys that Politician, Enterpreneur, Lawyer, Manager, CountryLegalRepresentative, and Journalist are non-disjoint sub-classes of Person. They inherit firstName and lastName from Person, as well as the fact that each instance livesIn in a City. All such properties are mandatory and functional, as indicated by the (1,1) labels in the diagram. In addition, each Manager also has a role (just one), and a CountryLegalRepresentative has an institutionalRole (just one). A Location can be a Country, Region, or City. Each Location has a name. Organizations are instead partitioned into GovernamentalOrganizations, Cooperatives, CriminalOrganizations, NewspaperOrganizations, Company(ie)s, and Party(ie)s. An Agency is a special type of GovernamentalOrganization. Each Organization has exactly one name. An Agency can investigatesOn Persons, whereas a Person can manage Cooperatives, or can be a component of (cf. isComponentOf) a CriminalOrganization. A Politician is enrolledIn just one Party, a Manager worksFor just one Company and a Journalist writesFor NewspaperOrganizations. All object properties are typed in both the domain and range (indicated by the arrows denoting inclusions from the domain or the range of the property to the class it is typed on), which means that no individuals other the ones among the instances of the typing classes can instantiate the domain and the range of the object property. Similarly, for the domain of data properties (whereas for simplicity we do not indicate the type of the values used for each attribute, that is we do not specify which value-domain the range of an attribute is included in).

4.3. Gazetteer

As discussed in Section 3, a Gazetteer annotates the documents on the basis of a set of lists containing names of entities. Such lists are used to find occurrences of the names in text (e.g., such for the task of NER). In our project, it was crucial to create all the Gazetteer lists used for document annotation, since all the articles that were analyzed were written in Italian, and when we started our project no reusable support for Gazetteer lists for Italian was available. To create the lists related to the domain entities of 'Mafia Capitale', two methods were used:

- The first based on *Human Knowledge*. This approach is used to write the lists containing words that identify certain categories to which the entities belong to, e.g., that contain all the words that identify lawyers. In order to create this list, we read some articles from newspapers about the domain of interest to better understand the terms identifying a specific category, as 'Lawyer', 'Advocate', 'Barrister' and 'Attorney' for the category of 'Lawyer'. Then we wrote the terms in the Gazetteer lists in both uppercase and lowercase letters, as GATE has a case-sensitive approach.
- The second based on *Open Data Sources*. This approach is used to find the first and last name of individuals belonging to a specific category, e.g., the members of the City Council of Rome, with the information of their roles as well. With this aim in mind, we downloaded specific lists of interest from open data sites, e.g., OpenPolitici [*] for the lists of Politicians currently in charge, DBpedia [†], etc.

The construction of such lists was demanding and required a significant amount of time to obtain good results. In our case, the effort was quantifiable in 7 person-days. This took into account a first phase of search of the information on the Web and a subsequent cleaning, which has lead to a translation and transformation of the extracted resources into the format of Gazetteers accepted by GATE. The value of 7 person-days strictly depends on both the language and the domain of interest. A richer ontology in a less widely spoken language could lead to a greater search effort and also to a greater manual writing of resources.

4.4. JAPE Rules in the Semantic Annotation Phase

The JAPE Transducer completes the semantic annotation phase, importing the user-written JAPE rules, used to create or modify text annotations on documents, into the GATE platform. JAPE allows users to recognize regular expressions in annotations on documents by utilizing the annotations previously produced from other PRs.

[*] `http://politici.openpolis.it/`
[†] `https://wiki.dbpedia.org`

The process of building annotations through JAPE rules is now being described. We have created many JAPE rules to annotate different entities based on the respective classes of the reference ontology. As an example, in the following we will discuss this process for the annotation *Lawyer*. For the annotation process we search for text expressions, *(i)* referring to a legal profession, and *(ii)* containing the first name and last name of a person qualified with the term under investigation. As an example, if we want to annotate as *Lawyer* the phrase ‘Carlo Taormina’ included in the following expression “L’avvocato Carlo Taormina ha difeso l’imputato davanti alla Corte” [*], the first step is to create a *Person* annotation from the attributes *first name* and *last name*. Since it is very difficult to create a list of all possible last names, we can not use a Gazetteer list to add such an annotation. Nevertheless, it is possible to write a Gazetteer list containing all the most common first names. Accordingly, we have created a Gazetteer list that produces a *Lookup* annotation when there is a match between an entry in the *first name* list and a word in the analyzed texts. Moreover, starting from the assumption that a first name is usually followed by a last name with a capital letter, we used the annotation *Token* created by the Unicode Tokenizer that has an orth-attribute indicating if a word starts with a capital letter. So we distinguished six cases for the construct *first name* and *last name* which are the common cases to be considered for the Italian language :

1. FirstName LastName, as in Antonio Bianchi;
2. FirstName SecondName LastName, as in Stefano Andrea Bianchi;
3. FirstName FirstLastName SecondLastName, as in Dante Delli Priscoli;
4. FirstName SecondName FirstLastName SecondLastName, as in Marco Antonio Della Valle;
5. FirstName FirstLastName ’ SecondLastName, as in Riccardo Dell’Aquila (note the apostrophe that separates the two parts of the last name);
6. FirstName SecondName FirstLastName ’ SecondLastName as in Alessandro Maria Dell’Acqua.

Once *Person* are in annotations on the text , we can use them to find the lawyers, by searching the sentence with annotations *Person* near an expression that contains a reference to the legal profession. This annotation is produced by the Lawyer JAPE rules [†], which we report below.

```
Phase: Lawyer// the name of the Phase
Input: Token Lookup Person //they are the input annotations of the
    Phase
Options: control = Brill
// indicates the kind of matching. In this case we use Brill: If one
    or more rules identify a match in the same portion of the document,
    they are performed all at once.

Rule:FindLawyerCaseOne
// indicates the name of the first rule. In a Phase we can have many
    rules.
((({Lookup.minorType==lawyer}):jobOne //is a word that is in
    Lawyer.lst (list of terms that refer to the profession of a lawyer)
    of annotated by Gazetteer; it is a word that identifies the
    lawyer’s job and we add a label jobOne on it
({Person}):personWorkOne) //is an annotation Person made up of First
    Name and Last Name found by the JAPE rule Person described above
    and we add a label personWorkOne
):personWork // is the label of the entire regular expression that
    made by the construct (Lookup)(Person)
```

[*]The translation of the sentence in English is “The lawyer Carlo Taormina has defended the accused in front of the Court”.

[†]The Lawyer rule in JAPE is translated to English for the reader, but in our work the JAPE rules were actually written in Italian.

```
-->
// Here starts the RHS part
{gate.FeatureMap features = Factory.newFeatureMap();
// this command is used to create a new type of Annotation
gate.AnnotationSet firstNameSet =
    (gate.AnnotationSet)bindings.get("personWorkOne");
// Through this command we can take the collection of regular
    expressions found in the text that match the regular expression
    with label personWorkOne defined in LHS
gate.Annotation firstNameAnn =
    (gate.Annotation)firstNameSet.iterator().next();
// This command is used to take a single match of persons
features.putAll(firstNameAnn.getFeatures());
// this command is used to export all the attributes present in the
    matching annotation Person to the new Annotation Lawyer
features.put("class","Lawyer");
// attaches a new attribute class to the new annotation created with
    value the string value Lawyer that represents the specific concept
    of the reference ontology. This attribute is useful for the
    Ontology Population Phase
features.put("rule", "FindLawyerOne");
// attaches a new attribute rule to the new annotation created with
    value the string value FindLawyer that represents the name of the
    rule that finds the annotation
outputAS.add(firstNameSet.firstNode(), firstNameSet.lastNode(),
    "Lawyer",features);
// creates the new annotation 'Lawyer' and its related attributes
}

Rule:FindLawyerCaseTwo // it indicates the name of the second rule

((({Person}):personWorkTwo
// is an annotation Person made up of First Name and Last Name found
    by the JAPE rule Person described earlier and a label personWorkTwo
    on it was added
({Token.string == ","}) // is a comma that follows the first Person
    annotation
({Lookup.minorType==lawyer}):jobTwo)
// is a word that is in Lawyer.lst (list of terms that refer to the
    profession of a lawyer) annotated by Gazetteer; it is a word that
    identifies the lawyer's job and a label jobTwo was added to it
):personWork // is the label of the entire regular expression that is
    made up by the construct (Person) , (Lookup)
-->
//the description of the RHS in this case is equal to the description
    of the RHS of FindLawyerCaseOne
{
gate.FeatureMap features = Factory.newFeatureMap();
gate.AnnotationSet firstNameSetDue =
    (gate.AnnotationSet)bindings.get("personWorkTwo");
gate.Annotation firstNameAnnDue =
    (gate.Annotation)firstNameSetDue.iterator().next();
features.putAll(firstNameAnnDue.getFeatures());
features.put("class","Lawyer");
features.put("rule", "FindLawyerTwo");
outputAS.add(firstNameSetDue.firstNode(), firstNameSetDue.lastNode(),
    "Lawyer",features);}
```

This rule annotated *Lawyer* as the human in these two types of sentences:

- *Lookup (with minor type equals to lawyer) Person*, as in "L'avvocato Carlo Taormina ha difeso l'imputato"[*] – the rule is activated so we have an annotation *Lawyer* on the *Person* identified as Carlo Taormina.
- *Person, Lookup(with minor type equals to lawyer)*, as in "Carlo Taormina, avvocato di fama internazionale ..." [†] – the rule is activated so we have an annotation *Lawyer* on the *Person* identified as Carlo Taormina.

We decided to use a short range between the annotation *Person* and *Lookup* on the basis of an analysis on text documents. In particular, the following LHS part of the rule shows also there cannot be more than two *Token*s:

```
({Lookup.minorType==lawyer}):jobOne
// is a word that is in Lawyer.lst (list of terms that refers to the
    profession of a lawyer) annotated by Gazetteer; it is a word that
    identifies the lawyer's job and we add a label jobOne to it
(({Token})?) // a generic token that can be present
(({Token})?) // a generic token that can be present
({Person}):personalavorauno
// is an annotation Persona made up of Name and Last Name found by the
    JAPE rule Person described above and a label personWorkOne was
    added to it
```

Finally, *Lawyer* was annotated in a special case when the category lawyer is referred to two persons. Let us consider the sentence "Gli avvocati Carlo Taormina e Mario Rossi sono appena entrati in aula" [‡]. This states that both Carlo Taormina and Mario Rossi are lawyers, but there is only one word referring to the legal profession carried out by both individuals. In order to put a correct *Lawyer* annotation on Carlo Taormina and Mario Rossi, the following DoubleLawyer JAPE rule needs to be adopted. It produces a match if there is a word in the plural referring to the legal profession followed by two annotations Person separated by the conjunction "e" ("and" in English):

```
//we show only the Regular expression to identify this special case of
    Lawyer
Rule: DoubleLawyer
(
({Lookup.minorType==lawyerMultiple}):jobOne
// is a word that is in Lawyers.lst (list of terms that refers to the
    profession of a lawyer when in the plural case) annotated by
    Gazetteer
({Person}):lawyerOne
(({Token.kind == word,Token.string =="e"})
({Person}):lawyerTwo
)
):personJob // the regular expression finds the following case
    (Lookup) (Person) "e" (Person)
```

4.5. *Multi-Lingual Noun Phrase Extractor (MuNPEx)*

Section 3 has been introducing MuNPEx as a multi-lingual noun phrase extraction component. Currently, this component does not support Italian and it is strictly necessary in order to use OwlExporter in the next phase. We have implemented an Italian support for MuNPEx based on the previously described TreeTagger POS. We started from the standard classes of the MuNPEx French version, which were modified to adapt to TreeTagger parameter files used for Italian. For

[*]The translation of the sentence in English is "The lawyer Carlo Taormina defended the accused".
[†]The translation of the sentence in English is "Carlo Taormina, the internationally famous lawyer ...".
[‡]The translation of the sentence in English is "The lawyers Carlo Taormina and Mario Rossi have just arrived in the courtroom".

each detected noun phrase, an annotation “NP” was added to the document, which includes several features [22]:

- *DET*, the NP determiner;
- *MOD*, a list of NP modifiers;
- *HEAD*, the NP head noun;
- *Pronoun*, boolean value (true/false) indicating an NP pronoun;
- *MOD2*, NP modifiers that appear after the HEAD noun.

MuNPEx contains language-specific and language-independent files and it is implemented as a set of grammars running in a multi-phase JAPE Transducer. For the Italian version we have implemented the file `it-np main.jape` that contains the transducer definition with five phases:

- `it-np-parts.jape` contains the language-specific definitions for the determiner, modifier, and head slots;
- `np-entities.jape` is a language-independent file defining which named entities to use for the NP HEAD slot;
- `check.jape` is a language-independent file that handles the special cases of differences between the input and output of the transducer;
- `np.jape` is a language-independent file that constructs annotations from the constituents detected in the previous phases;
- `clean.jape` cleans up temporary annotations.

5. CASE STUDY: EVALUATIONS AND RESULTS

In this section, the evaluation of the approach previously described in Sections 3 and 4 is reported. To test our techniques we have used 2657 articles generated by the crawling phase. In order to estimate the quality of our approach, we used the evaluation metrics usually adopted in Ontology-based Information Extraction [25]. We carried out two different evaluations:

1. In the first one, we chose 10 documents, in a random way among those produced by the crawling phase, and asked a domain expert to annotate them according to the domain ontology that we specified. The manual annotations aimed to identifying all and only the instances of the ontology predicates that could be extracted by the documents (on the basis of document content and the knowledge of the expert). These documents were then processed through our GATE-based pipeline, and a comparison was made between the annotations produced by our pipeline and the manual annotations of the domain expert. In order to evaluate the quality of our results, we took into account the Correct Annotations (a.k.a. True Positive), that is, the annotations that have been identified by our pipeline and that turned out to be coherent with the manual annotations, the Spurious Annotations (a.k.a False Positive), that is, the annotations returned by our pipeline but that were indeed wrong when compared with the manual annotations, and the Missing Annotations (a.k.a. False Negative), that is, those annotations that our pipeline was not able to find. These three parameters allowed us to calculate [26]:

    - *Precision*, i.e., the fraction of the correct annotations over the total number of identified annotations. It is formally defined as:

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

    - *Recall*, i.e., is the fraction of the correct annotations over the total amount of annotations. It is formally defined as:

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

- *F-measure*, i.e., the harmonic mean of *Precision* and *Recall*. It is formally defined as:

$$\textit{F-measure} = \frac{Precision * Recall}{0.5 * (Precision + Recall)} \quad (3)$$

2. In the second evaluation, all the 2657 articles obtained by the crawling phase were processed in our pipeline. Since so many documents were analyzed, obviously no previous reference annotation could be made, and also checking Precision for all annotations was not possible. Instead, we used this experiment to test the impact of our approach on a great amount of documents. The interesting measures are thus the total number of class, object property and data property assertions. Besides these measurements, we also verified the Precision of our results on a small portion of the analyzed documents.

5.1. Performance Evaluation

The results of the above evaluations are shown in Tables II and III, respectively, where *Words* indicates the number of words contained in the documents, and *CA*, *OPA* and *DPA* denote the number of class, object property, and data property assertions extracted by our pipeline. In Table III, the execution time represents the running time of the entire pipeline in order to process 2657 documents. Both experiments were performed on a MacBook Pro 9.2, with Intel Core i7 2,90 GHz and 8GB RAM.

Table II. Evaluation #1

Words	Total Annotations	CA	OPA	Spourious Ann.	Missing Ann.	Precision	Recall	F-measure
6.084	414	404	10	11	98	97,3	78,7	87%

Table III. Evaluation #2

Words	CA	OPA	DPA	Exec.time (sec.)
1.285.290	38.452	419	99.145	1.948,88 (~30 mins)

5.2. Discussion

We first notice that in our evaluations we got excellent Precision and good Recall, together with a very high number of data property assertions, a good number of class assertions, but few object property assertions.

As for Precision, we point out that the errors we obtained are mainly due to the wrong annotations associated to the word “Marino”, which is both a city close to Rome and the last name of a former mayor of Rome[*]. Indeed, in our pipeline, disambiguation was done through the JAPE rules we had defined. These rules, however, were able to identify the correct annotations only when the last name is coupled in the text with the first name of the mayor, which was often not the case in the selected documents.

A way to mitigate the mentioned problem and increment precision would be to add two additional PRs to our pipeline, namely the *Pronomial Coreference* and the *OrtoMatcher*. In fact, the use of such PRs would have allowed us to even substantially increase the number of retrieved object property assertions, as shown in the examples discussed below.

[*]Ignazio Marino was the mayor of Rome at the time of the mentioned investigation. Therefore, his name is quoted many times in the news articles used for our case study.

- *Pronominal Coreference (PC)*. This PR provides an annotation on pronouns that refer to an already annotated entity. For example, in the sentence “Marco Rossi is a successful lawyer. He lives in Milan”, our current solution recognizes that an individual named Marco Rossi is a lawyer, and also that Milan denotes a city, but it is not able to understand that “He” refers to the individual previously annotated and to thus extract the object property assertion representing the fact that Marco Rossi lives in Milan. By using the PC we can, instead, obtain the missing annotation. GATE currently provides this PR only for English, and so it could not be directly added to our pipeline.
- *OrtoMatcher*. It provides an annotation on an entity that is denoted in the text with an abbreviation of an already annotated expression. For instance, in the sentence “Marco Rossi is a successful lawyer. Rossi lives in Milan”, our current solution is not able to understand that Marco Rossi lives in Milan, since the person’s first name is omitted. OrtoMatcher would be able, instead, to find this annotation. However, the OrthoMatcher provided by GATE needs quite an exhaustive list of abbreviations for each non-abbreviated denotation of an entity, similar to Gazetteer lists. This approach is clearly impossible to pursue for cases with several possible abbreviations (as for a person). In the current release of our approach, this PR is actually used only for the most common abbreviations for the names of some famous organizations in the world. A more extensive usage of such PR needs further investigation and will be dealt with in future work.

We would like to underline that the addition to our pipeline of the above PRs would also allow to increase the Recall in our experiments, since its current value is mainly caused by the object property assertions we were not able to recognize.

In the second evaluation, it was not our aim to measure Precision and Recall for all documents. Our main goal was to evaluate the impact of our approach on a huge text corpus, to have an evidence of the computation time it requires in real-world scenarios. Nonetheless, to get an idea of the quality of the result, the Precision was measured for 1% of the output data, for which we got a 94% value.

6. SIMPLIFYING THE GENERATION OF GAZETTEER LISTS: DESIGN AND DEVELOPMENT

As explained in Section 4.3, the construction of Gazetteer lists is, in general, quite demanding and requires a significant amount of time to obtain quality results. We, thus, investigated alternative methods for the generation of Gazetteer lists that might reduce the manual workload and might be more easily replicable in different domains, thus increasing the generality of our approach. With this in mind, we considered the knowledge base Wikidata as source for the production of Gazetteer lists.

The aim of this variant was twofold: *(i)* to demonstrate how flexible the pipeline was and how some of its phases could be modified and incrementally refined to be improved; *(ii)* to show how semantic technologies can be used internally to the pipeline to make some phases more effective.

Wikidata is a collaboratively edited knowledge base which organizes a great amount of data in a structured way according to a general reference ontology. It is an openly accessible resource that follows the Semantic Web standards for exporting, interconnecting and querying data. Our idea was to exploit such a resource to partially automate the process of the generation of Gazetteer lists. We, thus, downloaded a Wikidata RDF dump (specifically we used the March 3, 2017 version), and loaded it on a graph database management system with RDF/SPARQL support, named Blazegraph*. This enabled us to access Wikidata information through standard SPARQL queries.

Through this approach, it was possible to avoid the tedious and time-consuming development of ad-hoc solutions for each Gazetteer list required. Once the setup of the system was completed, we just needed to define and execute a set of SPARQL queries to obtain the lists of interest. Such queries

*https://www.blazegraph.com/

have to be specified taking into account the set of lists to populate, the Wikidata ontology structure, and its data model. In Wikidata, every resource is classified as *item*, and every item is associated with statements defining its properties. For example, the item *Democratic Party* is associated with the statement *Istance Of: Political Party*. To complete the picture, the structure of the *Democratic Party* item[*] is reported below:

```
Democratic Party (Q47563)
Instance of(31) Political Party(Q7278)
Country(P17) Italy(Q38)
```

To obtain a list of items characterized by the same features, it is sufficient to create a specific SPARQL query. For example, to obtain the list of the names of Italian political parties, it is possible to use the property `wdt:P31`, which represents *instance of* (corresponding to the predicate *rdf:type*), and the property `wdt:P279`, which represents *subclass of* (corresponding to the predicate *rdfs:subclassOf*), and to reference the class `wd:Q7278`, which represents *political party*. Finally we need to filter this statement through the property `wdt:P17`, which represents *country of belonging*, referencing the class `wd:Q7278`, which represents *Italy*. Below we can observe the exact structure of the SPARQL query that returns the names of Italian political parties from Wikidata:

```
SELECT ?element
WHERE {
?item wdt:P31/wdt:P279* wd:Q7278 .
?item wdt:P17 wd:Q38 .
?item rdfs:label ?element
}
```

The result of the SPARQL query is a CSV file that has to be simply renamed into a .lst format file to be processed by the Gazetteer.

We conclude this section by providing a qualitative evaluation of the effort required by the Wikidata approach. Table IV reports the costs in person-days for the various tasks that were performed in our experiment. The person in charge of the tasks for the Wikidata approach was different from the one manually writing the Gazetters in the previous approach, but both of them were equally skilled, i.e., MsC students who had equal good grades and who had attended the same courses. Both were already acquainted with ontologies but not with GATE or Wikidata.

Task	Person-days
System setup	0,5
Wikidata ontology comprehension and analysis	0,5
Query specification	0,5
Query execution	0,3

Table IV. Person effort for the generation of the Wikidata-based Gazetteer lists

The total effort in this case was of 1,8 person-days, to be compared with the 7 person-days the OS/HK approach took (cf. Section 4).

The task of system setup can be avoided using the SPARQL online endpoint[†] provided by the Wikidata Query Service if the queries to be executed do not exceed the execution time limit imposed by the system. This also always allows to use the most up-to-date data. Moreover, once the practitioner has learned the most common relations used within Wikidata and has understood the

[*]https://www.wikidata.org/wiki/Q47729
[†]https://query.wikidata.org/

structures of interest, future applications of this approach will no longer involve an in-depth analysis of the Wikidata ontology, thus allowing to save more effort/time. Regarding the task concerning queries specification, the time that occurred for the generation of our 12 queries, was 0,5 person-days. The queries are easy to write, and, as can be seen from the previous example, they have a standard structure with variations only with respect to the Wikidata class. Their execution time is low. In fact the time to perform was about 0,3 person-days (a few hours).

In the next section, the quality of the lists obtained through the Wikidata approach with respect to those obtaned with the OS/HK approach is tested, comparing the quality of annotations produced through them.

7. SIMPLIFYING GAZETTEER LISTS GENERATION: TESTS AND RESULTS

In this section, the Wikidata approach for the generation of Gazetteer lists, described in Section 6, is compared with the open source (OS) and human knowledge (HK) approach described in Section 4.

With this aim, five lists containing words identifying *Politicians*, *Journalists*, *First Names*, *Criminal Organizations* and *Political Parties*, respectively, were taken into consideration. Each list was produced and used in two versions, i.e., an OS/HK-based one and a Wikidata-based one. The comparison was made on 15 articles, randomly selected among the ones returned by the crawling phase described in Section 4.1, and following the same criteria of the first evaluation discussed in Section 5: manual annotation was performed by a domain expert; then, the articles were annotated through the Gazetteer component of our pipeline by using the two different sets of lists generated by the two approaches; finally, Precision, Recall and F-measure have been computed in the two cases by using the manual annotation of the expert as ground truth.

For the evaluation of our GATE pipeline, we analyzed the results both after the annotation by the Gazetteer PR and at the end of our entire pipeline.. The aim of the first check (evaluation #1) was to highlight the contribution given to the semantic annotation by the Gazetteer PR, feed either with the OS/HK-based lists or with the Wikidata-based ones, and thus without the ‘adjustments’ provided by the JAPE rules. Therefore, the results obtained in this case are produced by a pipeline made up only of the following PRs: Document Reset, Unicode Tokeniser, RegEx Sentence Splitter, TreeTagger POS and Gazetteer. Such results are given in Table V. We’d like to underline that the ‘reduced’ pipeline does not actually produce OWL assertions corresponding to the instances of the ontology but returns documents annotated by the PRs that have been used in the pipeline (and in particular containing the annotations obtained through the Gazzeteer lists). Thus, Precision, Recall and F-measure are computed with respect to the corresponding annotations that the domain expert made. In this table, the columns “Total Annotations”, “Spurious Ann.”, and “Missing Ann.” refer to the overall number of annotations produced by our pipeline, the number of false positives, and the number of false negatives, respectively.

In the second check (evaluation #2) we looked at the final results obtained after both the semantic annotation and the ontology population phases of our pipeline. In this case the aim was to measure the overall impact of substituting, in our entire extracting process, the OS/HK-based Gazetteer lists with the Wikidata-based ones. The results are given in Table VI, where Precision, Recall and F-measure are given with respect to the OWL assertions produced through the manual annotation of the domain expert. In this case, the column “OWL Assertions”, “Spurious OWL Ass.” and “Missing OWL Ass.” refer to the total number of OWL assertions returned by our pipeline, the number of false positives, and the number of false negatives, respectively. Furthermore, it is to be underlined that OWL assertions that were taken into consideration refer to instances of classes of the ontology. For example, the assertions considered in the first row are of the form `ClassAssertion(Person p)`, where `p` is an instance of the class Person (cf. Section 4.2) that the Gazetteer list *First Name Open Source*, together with the use of the JAPE rules (as described in Section 4), allowed to identify. Similarly, the assertions considered in other rows refer to the ontology classes CriminalOrganization, Politician, Journalist, and Party.

We first discuss the results given in Table V. We initially notice that the number of entries in the lists created with Wikidata is in total larger than the number of entries obtained through the OS/HK

Table V. Results of Evaluation #1

List	Entries	Total Annotations	Spurious Ann.	Missing Ann.	Precision	Recall	F-measure
First Names Open Source	8913	322	83	0	74,2%	100%	85,2%
First Names Wikidata	2517	262	31	5	88,1%	97,8%	92,7%
Criminal Organizations Human Knowledge	25	37	21	0	43,2%	100%	60,3%
Criminal Organizations Wikidata	68	20	18	14	10%	12,5%	11%
Criminal Organizations Wikidata Mod.	273	30	24	10	20 %	37,5%	26%
Politicians Open Source	1083	11	0	40	100%	21,6%	35,4%
Politicians Wikidata	7778	17	0	34	100%	33,3%	50%
Journalists Open Source	369	4	0	4	100%	50%	66,6%
Journalists Wikidata	3727	8	0	0	100%	100%	100%
Political Parties Open Source	131	23	0	2	100%	92%	95,8%
Political Parties Wikidata	447	4	0	21	100%	16%	27,6%
Political Parties Wikidata Mod.	1293	26	7	6	73%	76%	74,5%

Table VI. Results of Evaluation #2

List	OWL Assertions	Spurious OWL Ass.	Missing OWL Ass.	Precision	Recall	F-measure
First Names Open Source	313	5	59	98,4%	83,9%	90,5%
First Names Wikidata	305	4	66	98,6%	82,0	89,5%
Criminal Organizations Human Knowledge	16	0	0	100%	100%	100%
Criminal Organizations Wikidata	2	0	14	100%	12,5%	22,2%
Criminal Organizations Wikidata Mod.	9	1	8	88,8%	50%	64,0%
Politicians Open Source	22	0	29	100%	43,1%	60,2%
Politicians Wikidata	23	0	28	100%	45,1%	62,2%
Journalists Open Source	4	0	4	100%	50%	66,6%
Journalists Wikidata	8	0	0	100%	100%	100%
Political Parties open source	23	0	2	100%	92%	95,8%
Political Parties Wikidata	4	0	21	100%	16%	27,6%
Political Parties Wikidata Mod.	26	7	6	73%	76%	74,5%

approach. In particular, if we do not consider the list containing Italian first names, which is the only one for which the OS/HK approach outperforms Wikidata in terms of amount of returned entries, this number is on average 9 times larger for the Wikidata lists. On the other hand, the Wikidata-based lists contain various entries with typos and flaws. For example, the *Political Parties Wikidata* list contains the entry “““PSDI””” (with wrong additional quotation marks), which does not allow to correctly annotate the word PSDI (which is an acronym of an Italian political party). The presence of more errors in the Wikidata-based lists was somehow expected. Our experiments, however, show that the Precision value for annotations with the Wikidata lists is in all cases the same or better than the Precision obtained with the OS/HK lists, except for the Criminal Organizations list. We'd like to point out that the list in the OS/HK approach has been accurately produced through a manual process based on human knowledge, since, before putting Wikidata into the loop, we could not find any open data source providing such information in the form of a dictionary list. Nonetheless, we could not avoid some errors in annotations, since the names of certain organizations are inherently ambiguous and may easily lead to the production of false positives. In this respect, we could not expect this problem to be mitigated by using a list automatically extracted from Wikidata.

For several Wikidata lists, also Recall is close to the one measured for the corresponding OS/HK lists. However, for *Criminal Organizations* and *Political Parties* it was very low due to the fact that GATE is case sensitive in its processing and we hadn't considered this in our initial Gazetteer lists. The two lists were thus refined by adding the three versions for each entry: all uppercase letters, all lowercase letters, and all capitalized words. In Tables V and VI these new lists are denoted as *Wikidata Mod.* (modified). With this fix, the value of Recall for both the lists had become greater than the one of the original Wikidata lists. However, for the *Political Party Wikidata Mod.* list, this approach caused a decrease in Precision. This was due to the introduction of new entries that have more than one meaning, such as the word *si* (i.e., the lowercase version of SI, which is the acronym of an Italian political party). This word is actually used in Italian as a reflexive pronoun, but with the use of the modified lists it was sometimes wrongly annotated as a political party. We' like to underline that the *Political Party Wikidata list* (i.e., the non-modified one) does not contain the all lowercase version of the entries, and thus this problem did not arise by using the orignal version of this list. Despite the decrease of Precision, the modification of this list produced an increment of Recall, which reached 79%, and of F-measure, which increased from 26,7% to 74,5%, that, thus, largely compensated the loss of Precision.

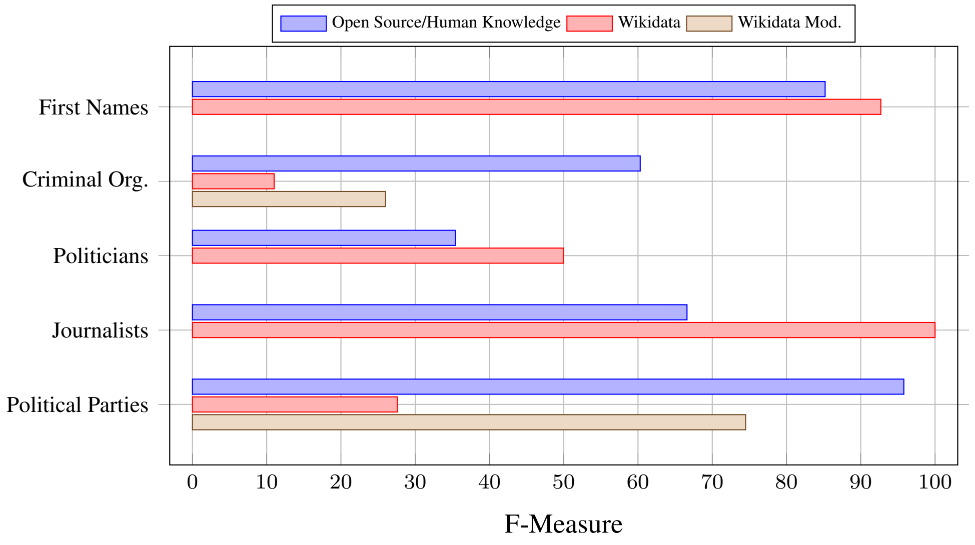


Figure 11. F-Measures for Evaluation #1

Wrapping up on evaluation #1, we can say that the overall results obtained with the Wikidata approach were quite good, as highlighted in Figure 11, where for the sake of presentation the values of the F-measure in the various cases in the form of a bar graph are reported.

The question now arises about what is the effect of executing our entire pipeline (that in particular means introducing in the annotation process the contribution of the JAPE rules we have specified). Let's now focus on the results given in Table VI. As it has been done for the previous evaluation, in Figure 12 we also show as a bar graph the F-measures given in this table. Comparing the F-measures of the two evaluations we soon notice that there is an improvement for the *First Names*, *Criminal Organizations* and *Politicians* lists, both in the OS/HK-based and in the Wikidata-based approach, whereas Precison, Recall, and thus F-measure remain the same as before for the other lists. This confirms that the use of JAPE rules, in general, improve the quality of the extracted information. In particular, for all versions of the *Criminal Organizations* list there was a great increase in Precision, which testifies that in this case the JAPE rules have primarily contributed to eliminate many of the spurious annotations introduced by the Gazetteer PR. A similar behaviour can be seen for both versions of the *First Names* lists. Instead, for the two versions of the *Politicians* list, the gain is on Recall (being Precision the same as in Table V), that is, in this case the JAPE rules have helped to identify relevant cases that were not captured in the previous evaluation. It can also be noticed that the improvement is almost always more substantial for the OS/HK-based approach. This is not surprising, since to some extent the JAPE rules have been specified to work on the annotations provided by the Gazetteer lists produced with the OS/HK-based approach. Nonetheless, the JAPE rules contributed to improve our results also in the case of Wikidata-based Gazetteer lists. Indeed, even if the OS/HK approach in evaluation #2 outperformed the Wikidata approach in 3 cases over 5 (thus at a first glance overturning the result of evaluation #1), overall the two approaches produced in fact very similar quality results, if we also take into account the number of annotations produced through each single Gazetteer list. This can be seen by computing the weighted average of the F-measures in the two cases, where the weight for the F-measure of each list is the fraction of the number of annotations obtained for this list over the total number of annotations, which returns 89,3% for the OS/HK approach and 86,4% for the Wikidata approach. Formally, the weighted average of the F-measures is:

$$\frac{\sum_{k=1}^{n} x_i \cdot w_i}{\sum_{k=1}^{n} w_i}$$

where each x_i is the F-measure of the i-th list, and its weight w_i is equal to $\frac{a_i}{\sum_{i=1}^{n} a_i}$, with a_i the number of annotations produced for the i-th Gazetteer list. For the sake of completeness, Table VII and Table VIII report the various x_i, w_i and the weighted F-measure.

Table VII. The weighted average of the F-measures for the OS/HK approach

List	x_i	w_i	**Wheighted F-Measure**
First Names Open Source	90.5%	0.82	75%
Criminal Organizations Human Knowledge	100%	0.042	4.23%
Politicians Open Source	60.2%	0.058	3.5%
Journalists Open Source	66.6%	0.01	0.7%
Political Parties Open Source	95.8%	0.06	5.8%
Total			89.2%

Table VIII. The weighted average of the F-measures for the Wikidata approach

List	x_i	w_i	**Wheighted F-Measure**
First Names Wikidata	89.5%	0.82	73.6%
Criminal Organizations Wikidata	64.0%	0.01	1.55%
Politicians Wikidata	62.2%	0.03	3.85%
Journalists Wikidata	100%	0.02	2.15%
Political Parties Wikidata	74.5%	0.07	5.22%
Total			86.4%

We conclude by having a look at the OWL data property assertions returned by our entire pipeline in the two approaches considered in this section. We'd like to underline that the OWLExporter

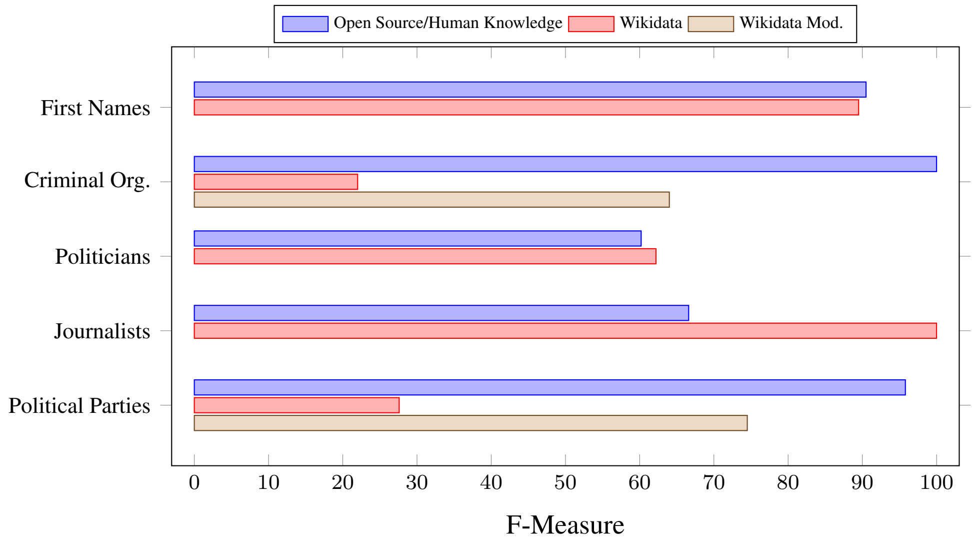


Figure 12. F-Measures for Evaluation #2

component often introduced several individuals (i.e., identifiers given in the form of IRIs*) to denote the same entity. In fact, the OWLExporter reconciled these individuals through `owl:sameAs` assertions, which say that two different IRIs are "equivalent". We can thus interpret them with the same object of the domain. For example, the OWL assertion `(A owl:sameAs B)` is indeed saying that `A` and `B` are two different identifiers for the same object (intuitively, they act as synonyms). As a consequence, every property of `A` is clearly also a property of `B`. Coming back to the OWL data property assertions returned by our pipeline, we notice that under the adoption of OS/HK-based Gazetteer lists a larger number of such assertions with respect to te ones acquired by making use of the Wikidata-based Gazetteer lists was obtained. This, however, does not really results in a loss of this kind of assertions. Actually, in the first approach data properties are, in fact, returned in a "closed form with respect to the `owl:sameAs` assertions", i.e., in the resulting OWL file, a data property for an individual `A` (e.g., `DataPropertyAssertion(name A ``Ignazio''))` is also explicitly associated to all the individuals related to `A` through `owl:sameAs` (e.g., also `DataPropertyAssertion(name B ``Ignazio'')` is present in the OWL file). Instead, this does not happen when Wikidata-based Gazetteer lists are used. This is mainly due to the way the lists are written down in files and the way JAPE rules use such annotations (it is to bear in mind that in our experiments the JAPE rules have been primarily designed to be coupled with the OS/HK-based Gazetteer lists).

To verify the behaviour mentioned above, and to show that it is, indeed, possible and easy to recover data property annotations missing from the OWL file produced by the pipeline equipped with the Wikidata-based Gazetteer list, we resorted to the use of an OWL reasoner, and used it to generate the data property assertions that were implicit in the OWL file (i.e., not asserted but implied by the ontology according to the `owl:sameAs` relation). We did so by considering only the two Gazetteer lists *Politicians Wikidata* and *Journalists Wikidata*, since they were the ones for which the most substantial (apparent) loss of data properties had been measured. The results of this further evaluation are given in Table VI, where "DPA", "DPM", and "DPAR" refer to the DataProperty assertions returned by our pipeline, those missed by our pipeline, and those obtained after the use of the OWL reasoner, respectively. For the sake of completeness, the number of Class assertions ("CA") computed by the pipeline are also reported. As shown in the table, in both cases the reasoner allowed us to recover all the missing assertions. As OWL reasoner we used HermiT [27], through the plugin available for Protégé[†]. To obtain the missing assertions it was sufficient to open our populated ontology in Protégé, start the reasoner and then export the result. We'd like to point out that HermiT computed the missing assertions in a few seconds (this was indeed an easy task, which essentially amounted to compute the transitive closure of the `owl:sameAs` relation, which is PTIME in the number of assertions).

Table IX. Results of evaluation #3

Lists	**CA**	**DPA**	**DPM**	**DPAR**	**Recover**
Politicians Wikidata	23	21	32	53	100%
Journalists Wikidata	8	0	16	16	100%

Realistically, we would obtain results similar to those commented in this section when the number of analyzed articles increases. We believe that this confirms that resorting to Wikidata for producing lists to be used by the Gazetteer PR produces annotations of good quality and is time saving (as said in Section 6, in our tests, 1,8 person-days were needed for the generation of the Gazetteer lists thorough Wikidata, instead of the 7 person-days required by the OS/HD approach).

*`https://www.w3.org/TR/owl2-syntax/`
[†]`https://protege.stanford.edu/`

8. CONCLUDING REMARKS

The automatic ontology population from raw texts is a very useful procedure, as it extracts data from documents that typically contain inherently irregular and potentially ambiguous information and assign such data with a precise structure and semantics. It is realized through a process that enriches the text with semantic annotations that lead to the identification of relevant entities to be finally exported into a reference ontology. Once this process is concluded, the user can access the data through the ontology, e.g., via query answering services, to obtain specific information also by leveraging reasoning abilities that ontologies allow for. In this respect, we point out that, though reasoning in expressive ontology languages may be computationally costly [4], light ontology languages exist, and they are specifically tailored to reason over data-intensive ontologies [28]. In these languages, for instance, query answering has the same computational cost of query evaluation over relational databases, and optimized techniques have been proposed that reduce such task to a standard SQL evaluation [29, 30, 31]. These results have pushed the use of ontologies for data management and access, and several tools nowadays exist for such purposes [32, 33, 34], as well as ontology-based data access techniques that have been adopted in several industrial contexts when large amount of data are being taken into consideration [35, 5]. In parallel, W3C has standardized three OWL tractable profiles, i.e., OWL fragments for which reasoning tasks such as ontology satisfiability, classification or even query answering, are tractable. The ontology that was used in our case is specified in one of such profiles, namely OWL 2 QL [36].

Through the automatic ontology population we can process very large amounts of documents in reasonable time, as testified by our experiments (cf. Table III). This ability goes far beyond the possibilities of human readers, which are obviously unable to analyze documents at the same rate of a machine. Our experiments also show (cf., Table II) that we were able to reach good levels of precision and recall, and so the quality of the results with our approach was not far from one obtained by manual annotations that a domain expert is able to make (on a limited number of documents). Thus, the trade-off between reachable quality and processable amount of documents makes automatic ontology population a powerful tool for structuring and interpreting information contained in free texts. This is especially useful in OSINT applications, where Web crawling processes return a great amount documents, which realistically cannot not be manually processed.

In this work we have proposed a working pipeline based on GATE that allows to populate a reference ontology from Italian text documents through the use of specific PRs. We have discussed an initial pipeline which consists of the following main steps:

- *Web Crawling* which is not specific to our approach and for which any standard technique can be applied.
- *Ontology Definition* which, as web crawling, is not specific to our approach, and for which ordinary methods and best practices for ontology and conceptual modeling can be adopted (cf., e.g., [37]). We require the ontology to be specified in the standard OWL syntax.
- *Semantic Document Annotation via GATE.* This paper precisely reports how this activity should be carried out, by explaining the specific GATE configuration that had to be created for Information Extraction from documents written in Italian. Notably, tasks similar to those we went through have to be executed in the practice when dealing with documents not written in English. Indeed, in these cases, it might not be effective enough to use standard GATE PRs, and it is obviously not possible to resort to the existing English-tailored resources for Information Extraction, such as available Gazetteer lists or the JAPE rules commonly used for English. In this respect, the value of our contribution is twofold, on the one hand, the process we set up and described is the same to be followed for other Western languages, and, therefore, we have provided a reusable tutorial on how to use GATE in these cases. On the other hand, the Gazetteer lists and JAPE rules we created for our pipeline are reusable resources to be exploited also in other domains where the text to be annotated is in Italian. We recall that many technical details of our approach, particularly useful for practitioners and for the reproducibility of our solution, are available through online resources accessible at `https://tinyurl.com/Ontology-population-via-GATE`. For the sake of

completeness we provide below a quick reference list of the main GATE components that were adopted for semantic annotations. In particular, in our initial pipeline we used:

- GATE Unicode Tokeniser, i.e, the standard GATE PR to split documents in tokens;
- RegEx Sentence Splitter, an advanced GATE PR to divide documents in sentences, which outperforms other standard GATE components in working with Romance and Germanic languages different from English;
- TreeTagger POS, a multi-language third party PR for document annotation with Part-Of-Speech and lemma information;
- Gazetteer, i.e., the GATE PR for recognizing entities and annotating them, which, as it has been stated, has been run with dictionaries specifically constructed for Italian language;
- JAPE Transducer, i.e., the GATE PR to add additional semantic annotations according to the JAPE rules it takes as input to the documents, which, similarly as the Gazetteer PR, has been executed with rules tailored to the Italian language.

• *Ontology Population*. We have shown how the ontology is finally populated with instances extracted from semantic annotated documents and exported in OWL. In particular, we have shown how to customize the OwlExporter component, a third party PR to be used inn Italian text documents in order to make the ontology population phase easier.

After presenting the pipeline above, we have also discussed a second *partially-automated* pipeline, in which an available open-knowledge base (namely, Wikidata) has been exploited to simplify the generation of Gazzetter lists and provide a more definite procedure to approach their definition (thus increasing the replicability of our solution). This second pipeline consists of the same phases of the previous one, and makes use of exactly the same components. However, the generation of the Gazetteer lists sub-task faces a completely different approach based on querying through the standard SPARQL language, an open ontology available on the Web, i.e., Wikidata, in order to create such lists without a specific design (as done in the first pipeline). We have shown through additional experiments that such an approach produces quality results that are comparable to the ones reached in the previous pipeline (cf. Table VI), where the Gazzetteer lists are produced through a less structured approach that requires more manual effort. We have also discussed workload saving allowed by the use of Wikidata for the generation of the Gazzetteer lists.

The pipelines proposed in this paper can be easily replicated over different domains within the Italian language. Indeed, for this language the configuration of the GATE PRs to be adopted is exactly the same that has been used for our case study. Of course, the Web Crawling and Ontology Definition phases will be different, since they are domain specific (but it is worthwhile mentioning that for many application domains there are standard ontologies that can be easily reused or adapted). Also, even though the Gazetteer lists and the JAPE rules that have been produced for Italian can be also good for other projects, in general dictionaries and JAPE rules tailored to the specific domain at hand need to be produced to increment the quality of the extraction process. As it has been stated, the costs related to the generation of the Gazetteer lists are reduced in our second pipeline (we have discussed in the paper the effort required for the various tasks and how it can be reduced in the second pipeline).

The value of our work goes, however, beyond the application to Italian. Actually, the detailed description we provide on the development of the pipeline, and of its specific PRs, makes it a reference guideline for similar components to be developed by practitioners for Information Extraction from documents written in other languages. It is true that the GATE components that are needed for the various extraction tasks are available mainly for English, and their application to other languages requires a complex activity that has been discussed in this paper. Our approach is indeed language-independent, and therefore easily generalizable.

Further improvements to our approach include, as discussed in Section 5.2, the addition in our pipeline of the Pronominal Coreference and the OrtoMatcher components. Also, we are working on further refinements of our approach in order to increase the level of automation of (some of) the phases that now require manual work. In particular, we are currently investigating a way to automatize the definition of JAPE rules. In this respect, a promising direction we are following is the exploitation of the reasoning abilities provided by the ontology to generate new JAPE rules starting from a nucleus of manually-specified JAPE rules. For example, given a JAPE rule J, we can easily produce replicas of J where each annotation in the rule body, referring to a named class of the ontology, is substituted with all its subclasses (computed by reasoning over the ontology), in a way that the effect of J is, in fact, obtained even in those cases in which a subclass is used in an annotation. A similar idea also can be applied to Gazzetteer lists to automatically increment the entries in dictionaries by exploiting the ontology hierarchies. W would like to mention that, in the OWL 2 QL language used in our case study, the deductive closure of the ontology which allows to obtain all subclasses of a given class, is finite and can be computed in polynomial time.

ACKNOWLEDGEMENT

This work has been partly supported by the Leonardo Company (formerly Selex ES) in the context of the *XASMOS* program, by the Italian project *RoMA* (SCN_00064) and by the initiative *Filiera Sicura*. The work by Giulio Ganino has been sponsored by the FILAS grant *Laboratori teorico-sperimentali a supporto delle applicazioni spaziali delle industrie laziali* (FILAS-RU-2014-1058).

REFERENCES

1. Baldoni R, De Nicola R. The white book on cyber-security 2015. URL `https://www.consorzio-cini.it/index.php/it/labcs-home/libro-bianco`.
2. Scannapieco M, Barcaroli G, Summa D, Scarnò M. Using internet as a data source for official statistics: a comparative analysis of web scraping technologies. *Proc. of NTTS'15*, 2015. URL `https://ec.europa.eu/eurostat/cros/system/files/Barcaroli-etal_WebScraping_Final_unblinded.pdf`.
3. Guarino N. Formal ontology in information systems. *Proc. of the 1st Int. Conf. on Formal Ontology in Information Systems (FOIS 1998)*, Frontiers in Artificial Intelligence, IOS Press, 1998; 3–15.
4. Baader F, Calvanese D, McGuinness D, Nardi D, Patel-Schneider PF ((eds.)). *The Description Logic Handbook: Theory, Implementation and Applications*. 2nd edn., Cambridge University Press, 2007.
5. Antonioli N, Castanò F, Coletta S, Grossi S, Lembo D, Lenzerini M, Poggi A, Virardi E, Castracane P. Ontology-based data management for the Italian public debt. *Proc. of the 8th Int. Conf. on Formal Ontology in Information Systems (FOIS 2014)*, 2014; 372–385.
6. Collobert R, Weston J, Bottou L, Karlen M, Kavukcuoglu K, Kuksa P. Natural language processing (almost) from scratch. *J. of Machine Learning Research* Nov 2011; **12**:2493–2537. URL `http://dl.acm.org/citation.cfm?id=1953048.2078186`.
7. Cunningham H. *Developing Language Processing Components with GATE Version 8*. University of Sheffield Department of Computer Science, 2014.
8. Poibeau T, Saggion H, Piskorski J, Yangarber R ((eds.)). *Multi-source, multilingual information extraction and summarization*, Springer: Berlin; New York, 2013. URL `http://link.springer.com/book/10.1007/978-3-642-28569-1`.
9. Navigli R. Word sense disambiguation: a survey. *ACM Computing Surveys* 2009; **41**(2):1–69.
10. Kibble R. *Introduction to Natural Language Processing*. University of London, 2013.
11. Bird S, Klein E, Loper E. *Natural Language Processing with Python: Analyzing Text with the Natural Language Toolkit*. O'Reilly: Beijing, 2009, doi:http://my.safaribooksonline.com/9780596516499. URL `http://www.nltk.org/book`.
12. Johnson M, Khudanpur S, Ostendorf M, Rosenfeld R. *Mathematical Foundations of Speech and Language Processing*. Springer New York, 2004. URL `https://books.google.it/books?id=Zhp3DA0Ric8C`.
13. Zhao H, Zhang X, Kit C. Integrative semantic dependency parsing via efficient large-scale feature selection. *J. of Artificial Intelligence Research* 2013; **46**:203–233.
14. Cunningham H, Maynard D, Bontcheva K, Tablan V. GATE: A Framework and Graphical Development Environment for Robust NLP Tools and Applications. *Proc. of ACL'02*, 2002.
15. Cunningham H, Tablan V, Roberts A, Bontcheva K. Getting more out of biomedical documents with gate's full lifecycle open source text analytics. *PLoS Computational Biology* ; **9**(2), doi:10.1371/journal.pcbi.1002854.
16. Gruber TR. Towards principles for the design of ontologies used for knowledge sharing. *Formal Ontology in Conceptual Analysis and Knowledge Representation*, Guarino N, Poli R (eds.), Kluwer Academic Publishers, 1993.
17. Guarino N, Oberle D, Staab S. What is an ontology? *Handbook on Ontologies*. Springer, 2009; 1–17.
18. Lembo D, Pantaleone D, Santarelli V, Savo DF. Easy OWL drawing with the graphol visual ontology language. *Proc. of the 15th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2016)*, Baral C, Delgrande JP, Wolter F (eds.), AAAI Press, 2016; 573–576.
19. Lembo D, Pantaleone D, Santarelli V, Savo DF. Drawing OWL 2 ontologies with eddy the editor. *AI Commun.* 2018; **31**(1):97–113.
20. Schmid H. Probabilistic part-of-speech tagging using decision trees. *Proc. of the Int. Conf. on New Methods in Language Processing*, 1994; 44–49.

21. Petasis G, Karkaletsis V, Paliouras G, Krithara A, Zavitsanos E. Ontology population and enrichment: State of the art. *Knowledge-driven Multimedia Information Extraction and Ontology Evolution*, Paliouras G, Spyropoulos CD, Tsatsaronis G (eds.). Springer-Verlag: Berlin, Heidelberg, 2011; 134–166. URL `http://dl.acm.org/citation.cfm?id=2001069.2001075`.
22. Witte R. Multi-lingual noun phrase extractor(munpex). URL `https://github.com/SemanticSoftwareLab/TextMining-MuNPEx/blob/master/doc/munpex.pdf`.
23. Witte R, Khamis N, Rilling J. Flexible ontology population from text: The OwlExporter. *Proc. of LREC'10*, 2010.
24. Witte R. OwlExporter guide for users and developers 2014. URL `http://www.semanticsoftware.info/system/files/owlexporter-3.2.pdf`.
25. Maynard D, Li Y, Peters W. NLP techniques for term extraction and ontology population. *Ontology Learning and Population: Bridging the Gap between Text and Knowledge*. 2008; 107–127.
26. Maynard D, Li Y, Peters W. Metrics for evaluation of ontology-based information. *Proc. of WWW 2006 Workshop on Evaluation of Ontologies for the Web*, 2006.
27. Glimm B, Horrocks I, Motik B, Stoilos G, Wang Z. Hermit: An owl 2 reasoner. *J. of Automated Reasoning* 2014; **53**(3):245–269, doi:10.1007/s10817-014-9305-1.
28. Calvanese D, De Giacomo G, Lembo D, Lenzerini M, Rosati R. Tractable reasoning and efficient query answering in description logics: The *DL-Lite* family. *J. of Automated Reasoning* 2007; **39**(3):385–429.
29. Rosati R, Almatelli A. Improving query answering over *DL-Lite* ontologies. *Proc. of the 12th Int. Conf. on the Principles of Knowledge Representation and Reasoning (KR 2010)*, 2010; 290–300.
30. Venetis T, Stoilos G, Stamou GB. Query extensions and incremental query rewriting for OWL 2 QL ontologies. *J. on Data Semantics* 2014; **3**(1):1–23.
31. Pérez-Urbina H, Horrocks I, Motik B. Efficient query answering for OWL 2. *Proc. of the 8th Int. Semantic Web Conf. (ISWC 2009)*, *Lecture Notes in Computer Science*, vol. 5823, Springer, 2009; 489–504.
32. Civili C, Console M, De Giacomo G, Lembo D, Lenzerini M, Lepore L, Mancini R, Poggi A, Rosati R, Ruzzi M, *et al.*. MASTRO STUDIO: Managing ontology-based data access applications. *Proc. of the VLDB Endowment* 2013; **6**:1314–1317.
33. Calvanese D, Cogrel B, Komla-Ebri S, Kontchakov R, Lanti D, Rezk M, Rodriguez-Muro M, Xiao G. Ontop: Answering SPARQL queries over relational databases. *Semantic Web* 2017; **8**(3):471–487.
34. Nenov Y, Piro R, Motik B, Horrocks I, Wu Z, Banerjee J. Rdfox: A highly-scalable RDF store. *Proc. of the 14th Int. Semantic Web Conf. (ISWC 2015)*, 2015.
35. Kharlamov E, Hovland D, Skjæveland MG, Bilidas D, Jiménez-Ruiz E, Xiao G, Soylu A, Lanti D, Rezk M, Zheleznyakov D, *et al.*. Ontology based data access in statoil. *J. Web Sem.* 2017; **44**:3–36.
36. Motik B, Fokoue A, Horrocks I, Wu Z, Lutz C, Cuenca Grau B. OWL Web Ontology Language profiles. *W3C Recommendation*, World Wide Web Consortium Oct 2009. Available at `http://www.w3.org/TR/owl-profiles/`.
37. Staab S, Studer R ((eds.)). *Handbook on Ontologies*, Springer, 2009.