Semano: Semantic Annotation Framework for Natural Language Resources

David Berry[1] and Nadeschda Nikitina[2]

[1] QinetiQ, UK
[2] University of Oxford, UK

Abstract. In this paper, we present Semano — a generic framework for annotating natural language texts with entities of OWL 2 DL ontologies. Semano generalizes the mechanism of JAPE transducers that has been introduced within the General Architecture for Text Engineering (GATE) to enable modular development of annotation rule bases. The core of the Semano rule base model are rule templates called *japelates* and their instantiations. While Semano is generic and does not make assumptions about the document characteristics used within japelates, it provides several generic japelates that can serve as a starting point. Also, Semano provides a tool that can generate an initial rule base from an ontology. The generated rule base can be easily extended to meet the requirements of the application in question. In addition to its Java API, Semano includes two GUI components — a rule base editor and an annotation viewer. In combination with the default japelates and the rule generator, these GUI components can be used by domain experts that are not familiar with the technical details of the framework to set up a domain-specific annotator. In this paper, we introduce the rule base model of Semano, provide examples of adapting the rule base to meet particular application requirements and report our experience with applying Semano within the domain of nano technology.

1 Introduction

It is widely acknowledged that finding particular information within unstructured natural language documents is significantly harder than finding it within structured data sets. Despite the impressive state of the art in the area of intelligent information infrastructure, accessing information enclosed within natural language documents remains a big challenge. For instance, if we are looking for scientists married to politicians on the Web using the Google search engine, we get back results that cover in detail the topic of same-sex marriage. However, we do not get back a single document that mentions marriage between scientists and politicians. In contrast, if we look within documents that contain the relevant information in form of *semantic annotations* — markup indicating the meaning of document parts — we can find relevant results by means of structured queries.

In recent years, semantic annotations have significantly gained in popularity due to the Schema.org[1] initiative and semantic wikis such as Semantic MediaWiki [9]. Despite

[1] `https://schema.org/`

P. Mika et al. (Eds.) ISWC 2014, Part I, LNCS 8796, pp. 503–518, 2014.

the very recent introduction of Schema.org, around 30% of web pages crawled by Bing[2] early this year include semantic annotations [12].

While semantic annotations can be easily added to dynamically generated content on the Web, a considerable proportion of digital information is stored as natural language. For instance, PubMed – the online database of biomedical literature [20] – currently comprises over 23 million entries and continues growing at a rate of 1.5 publications per minute [18]. As a consequence, this information is inaccessible for a wide range of important applications.

In the recent years, considerable effort has been invested into research on extracting structured information from natural language resources. Numerous domain-specific and corpus-specific information extraction (*IE*) systems have been developed in the past to give rise to valuable, high-quality data sets such as Yago [17], DBpedia [4] and a large part of the Freebase [5] data set. These data sets are being intensively used by the community for a wide range of applications. For instance, the latter is being used by Google to enhance certain search results.

While those IE systems demonstrate that extracting high-quality structured data from unstructured or semi-structured corpora is possible in principle, the development of such systems involves a substantial amount of work. Recently, numerous tools have emerged facilitating the development of IE systems. Among them is *General Architecture for Text Engineering (GATE)* [6] – an open source framework for the development of IE systems. GATE has over 100,000 users[3] and around 70 plugable components. GATE supports the development of systems that *annotate* documents — recording specific characteristics of text within a document. The focus of this framework are *ontology-based semantic annotators* — systems that identify and record occurrences of ontology entities within documents.

At the core of the GATE framework are *JAPE transducers* — generic annotators that manipulate annotations within documents in accordance with a set of annotation rules provided by user. We refer to this set of rules as a *rule base*. While the mechanism of JAPE transducers can notably speed-up the development of annotators by reducing the amount of code that needs to be written, it still takes a considerable amount of work to develop a comprehensive rule base. The reason for this is the rule base model of JAPE transducers which does not support modularity. In particular in case of ontology-based annotators, JAPE rule bases tend to contain a significant amount of redundancy. As rules usually undergo numerous revisions, modifications are very common in rule base development. Thus, the lack of modularity results in substantial cost.

In this paper, we present Semano — a framework that can significantly reduce the effort of rule base development by enabling modularity. Semano is a modular rule store that has been designed to efficiently support the development of ontology-based annotators. The core of Semano are *Japelates* — JAPE-style rule templates — and *Japelate instantiations* — statements that define actual annotation rules by binding Japelate parameters to concrete values. As we will discuss in our paper, representing a rule base as Japelates and their instantiations significantly reduces redundancy within the rule base. For instance, the Semano representation of our example rule base *NanOn* is by an order

[2] `https://www.bing.com/`

[3] `http://en.wikipedia.org/wiki/General_Architecture_for_Text_Engineering`

of magnitude smaller due to the reduced amount of redundancy and is notably easier to update in comparison to its representation as a JAPE rule base. A further improvement is achieved in Semano by introducing *abstract instantiations* — japelate instantiations that bind only the values of certain parameters and can be reused in other instantiations — and variable number of arguments in analogy to `varargs` in programming languages such as Java.

Another important contribution of Semano to the community of GATE users is a module for accessing OWL 2 DL ontologies within GATE. Currently, GATE provides only a rudimentary support for ontology-based annotation. The built-in ontology module is based on OWLIM [3], which supports RDF(S) and a fragment of OWL Lite. This is a notable limitation as many domain-specific ontologies make use of more expressive ontology languages. Semano overcomes this limitation and introduces support for OWL 2 DL ontologies. Since many annotation rules rely on class hierarchies within ontologies, Semano can classify ontologies prior to document annotation. For this task, Semano has a choice of six different built-in reasoners. This is an important feature, as different reasoners work best for different fragments of the OWL 2 DL ontology language [2].

Further, Semano provides two GUI components — a rule base editor and an annotation viewer. They build on the GATE framework and enable an efficient use of a Semano rule base within GATE-based applications. The rule base editor provides a convenient way of exploring and extending the rule base. Its core features are assistance in creating japelate instantiations and flexible rule filters. The annotation viewer enables engineers to efficiently test a rule base on a particular document and update particular rules. In general, the annotation viewer can serve as a platform for evaluating the quality of generated annotations and building up a training corpus, which is otherwise significantly more work.

The paper is structured as follows. In Section 2, we discuss the impact of ontologies on the modularity of annotators and give a brief overview of Semano. In Section 3, we present the Semano rule base model and discuss its role in modular rule base development. In Section 4, we demonstrate how Semano's default features can be extended to meet application-specific requirements. Section 5 outlines the annotation of corpora with Semano and reports our experience of annotating a corpus with the NanOn rule base. In Section 6, we give an overview of the annotation viewer and the rule base editor before concluding in Section 7.

The Semano framework including example japelates and a test document is available at https://github.com/naditina/gate-semano. A demo video showing the main features of Semano is also available[4].

2 Supporting Ontology-Based Annotation with Semano

Semano has been developed within the context of a project called *NanOn*. The aim of NanOn was to annotate scientific literature within the domain of nano technology in order to make literature search more efficient. Over the course of the project, we found that ontologies bring certain important advantages for document annotation as opposed

[4] `https://www.youtube.com/watch?v=nunxWXgWcBU&feature=youtu.be`

to a simple set of semantic labels. In particular, we found the following benefits to be very prominent:

Class Hierarchies: We can use subsumption relationships between classes encoded within the ontology instead of repeating this information within annotation rules and annotated documents. If, for instance, the ontology includes the information that a `Scientist` is a `Person`, and we find an occurrence of a `Scientist` in a document, then we know that this is also an occurrence of `Person`. Thus, we do not need to include the second annotation into our annotated document. We also do not need to include rules for identifying occurrences of scientists into the set of rules for identifying occurrences of a person. Thus, systems that rely on class hierarchies within ontologies are more modular.

Domain and Range Restrictions: Detecting relations between entities within documents is a very difficult task. While classes often occur within a document as specific expressions, e.g., `Indium Tin Oxide` or `ITO`, the indication for a relation between entities is typically much more subtle and requires a detailed analysis of the corresponding document part. Among other things, the precision of relation annotation can be significantly improved by specifying which types of entities can be related to each other. In ontologies, this information can be modelled in an natural way as domain and range restrictions. In NanOn, we found the domain and range restrictions from the ontology to be indispensable indicators for the corresponding relations in documents. For example, the relation `materialProperyOf` within the expression *conductivity of ITO* is easy to detect, but only if we know that `conductivity` is a `MaterialPropery` and *ITO* is a `Material` and that `materialProperyOf` typically connects entities of type `MaterialPropery` and `Material`. In combination with class hierarchies, domain and range restrictions in ontologies allow us to significantly simplify the development of annotation rules and foster modularity of IE systems.

Within the NanOn project, an ontology specified in the Web Ontology Language OWL 2 DL[13] modelling the scientific domain of nano technology has been used to automatically annotate scientific publications. Within this context, the Semano framework has been developed to specifically address the needs of ontology-based annotation. We found that Semano has helped us to boost the productivity over the course of rule base development. The core functionality of Semano is accessible via a Java API as well as through a GUI application loadable within GATE and includes the following:

Generating a Generic Rule Base from an Ontology: To speed-up the development of a rule base, Semano can generate an initial rule base from an ontology that can serve as a starting point and can be refined into more specific annotation rules. For this purpose, we use the information within the ontology such as class names, labels as well as domain and range restrictions. This features can easily be customized or extended to be based on further information sources.

Accessing and Manipulating a Rule Base: The Semano Java API includes common manipulation operations for loading, exploring and updating a rule base. All these operations are also available within the Semano rule base editor. It is further possible to export a rule base in JAPE format for further processing in GATE.

Loading and Classifying OWL 2 DL Ontologies: To enable the use of OWL 2 DL ontologies in GATE-based applications, Semano provides a GATE module that is based on the OWL API [7] and supports numerous formats[5]. Semano enables the user to choose a reasoner for classifying a particular ontology. This is important as, depending on the ontology, the right choice of the reasoner has a notable impact on the overall performance. For instance, in order to use Snomed CT [16] as a basis for annotation, Snorocket [10] or ELK [8] have to be selected as other reasoners fail to classify it. However, for ontologies making use of further OWL 2 DL features, these reasoners might miss some subsumption relationships as they have not been designed to deal with the entire set of OWL 2 DL features. Currently, Semano provides access to six reasoners that can be used for document annotation including HermiT [14], FacT++ [19], Pellet [15], MoRe [1], Snorocket [10] and ELK [8]. In general, Semano can be easily extended to include further reasoners that implement the corresponding Reasoner interface in OWL API.

Annotating Corpora with a Rule Base: Semano can be used to annotate documents or entire corpora with given a rule base and an OWL ontology. Semano builds on JAPE transducers. It translates the rule base into a JAPE rule base before instantiating a JAPE transducer. Per default, Semano saves annotated documents in GATE format. However, it can also export annotations as ontology instances, triples or quads.

GUI Components: Semano includes two GUI components — a rule base editor and an annotation viewer. The former provides convenient means of accessing and updating the rule base, while the annotation viewer is designed to efficiently support an evaluation of generated annotations in documents.

In the subsequent sections, we elaborate on selected features of Semano. In particular, we discuss the modularity of Semano rule bases, the two GUI components and demonstrate how Semano can be used to implement rules that meet application-specific requirements.

3 Achieving Modularity with the Semano Rule Base Model

The rule base model of Semano builds on the framework of *JAPE transducers* and provides an abstraction layer to add support for modularity. Like JAPE rules, Semano rules operate on annotations enclosed within documents and their characteristics called *features*. The richer the information provided within these annotations, the more information can be accessed within Semano rules. In our examples within this section, we assume that documents have been pre-processed using an English tokenizer, a sentence splitter, a part of speech (POS) tagger and an orthoMatcher, which are all included within the standard GATE application *ANNIE*. Our examples refer to annotations of type `Token`, `Sentence` and `Mention`. The former annotate distinctive words within the document with certain characteristics, from which we use *category* — the POS category of the token, e.g., NN for noun and JJ for adjective, and *string* — the actual value of

[5] `http://owlapi.sourceforge.net/index.html`

```
1  Phase: metaPropertyOf
2  Input: Mention
3  Options: control = appelt debug = true
4
5  Rule: rule113
6  (
7        ({Mention.class=="Meta_property"}):domain
8        ({Mention.class=="Material_property"}):range
9  ):binding
10 -->
11 {  ... 40 more lines of  Java code
12          to create an annotation ...
13 }
```

Fig. 1. JAPE rule for the ontology relation *metaPropertyOf*

the token within the document. Annotations of type `Mention` are generated by Semano and include arbitrary features set by annotation rules. We use the feature *class*, which refers to the ontology entity occurring in the annotated document part.

We now discuss the differences within the rule base model of JAPE transducers and Semano. To this end, we briefly introduce the former before elaborating on the latter. Annotation rules passed to JAPE transducers are defined within scripts written in a language specifically introduced for programming JAPE transducers, and Java. An example of a JAPE script is given in Fig. 1. A JAPE script typically consists of a header (lines 1-3) and a set of rule definitions (line 5 onwards). In this example, we have only one rule — rule113 — that annotates documents with the ontology relation `metaPropertyOf`. Each such rule definition in turn consists of a rule *body* (lines 6-9) — a JAPE-style regular expression over an annotation set — and a *head* (line 11 onwards) — code that will be executed when the rule body matches a certain part of the document. In Fig. 1, the rule body matches two consecutive annotations of type `Mention` that record an occurrence of the ontology classes `Meta_property` and `Material_property`, respectively. The rule head is Java code that creates an annotation of type `Mention` and sets its features.

While the mechanism of JAPE transducers enables a development of sophisticated annotation rules, it is not suitable as a basis for large rule bases. The reason for this is the poor support for modularity, which makes the development of annotation rules unnecessarily cumbersome. For instance, if we consider the JAPE script given in Fig. 1 and examine other rules developed for NanOn, we notice that 18 rule definitions within the NanOn rule base are identical except for the values in blue. The most frequently shared rule structure is used in NanOn in over 16,000 different rules. An even larger number of rules share a large proportion of the code within the rule head. Thus, while the functionality provided by transducers is highly valuable for the development of IE applications, the mechanism of JAPE scripts in its current form leads to a significant amount of redundancy in rule specifications. Since extension and modification of rule specifications are rather common in IE development, the problem needs to be addressed in order to avoid a substantial overhead. In the following, we discuss the mechanisms developed within Semano to achieve higher modularity.

3.1 Introducing Parameters

The first step taken in Semano to increase modularity of rule bases is introducing parameters for rules analogously to parameters of methods which are typical in programming languages. Semano introduces *japelates* — rule templates that define the basic structure of an annotation rule based on a set of parameters. At a later point, a japelate can be *instantiated* to form a concrete rule by specifying actual values for the parameters declared within the japelate. For instance, the JAPE script given in Fig. 1 can be transformed into a japelate `consecutiveDomainRangeMatcher` that accepts parameters for the values in blue, i.e., the rule name and two class IRIs. Additionally, we introduce parameters for the ontology IRI and the relation IRI, which are used within the rule header. The japelate `consecutiveDomainRangeMatcher` is shown in Fig.2. Based on this japelate, can then obtain a definition of `rule113` by instantiating it as follows:

```
1  JAPELATE PARAMETERS:
2  0: LITERAL, Rule ID
3  1: LITERAL, Ontology IRI
4  2: ENTITY,  Relation IRI
5  3: ENTITY,  Domain class IRI
6  4: ENTITY,  Range class IRI
7
8
9  JAPELATE BODY:
10 Rule: $0$
11 (
12       ({Mention.class=="$3$"}):domain
13       ({Mention.class=="$4$"}):range
14 ):binding
15 -->
16 { ... Java code referring to $1$ and $2$ ... }
```

Fig. 2. Japelate `consecutiveDomainRangeMatcher`

```
rule113: domainRangeBased(http://www.nanon.de/ontology/,
         metaPropertyOf, Meta_property, Material_property)
```

Within the NanOn rule base, the separation into japelates and rules was very effective for increasing modularity. Overall, over 17,000 rules instantiated from in total seven japelates encode the entire NanOn rule base, which otherwise would be represented by the same number of relatively long JAPE rules. As a result, the NanOn rule base is not only significantly smaller, but also notably easier to update. For instance, renaming an annotation feature that is being set within the Java code in the rule head (lines 11-13, Fig.1) will affect over 1,000 JAPE rules. In contrast, within a Semano rule base it affects only a few japelates and no rules.

3.2 Variable Number of Arguments

While introducing parameters is already a big step towards modular rule base development, the result can be further improved as demonstrated by the following example. We consider the rule head of the japelate `nonINMatcher`, which is a simplified version of the corresponding japelate within the NanOn rule base:

```
Rule: $0$
(
      {Token.string==~"$3$($4$)", Token.category!=IN}
):binding
```

This japelate can be instantiated, for instance, by setting 3 to *(?i)* and setting 4 to *absorbed|absorption*. This will match all annotations of type `Token` that are not a preposition (ensured by `Token.category!=IN`) and whose string value contains either *absorbed* or *absorption* ignoring the case. While this japelate can be used for all terms consisting of a single word that is not a preposition, we need a new japelate for terms consisting of n such words. For instance, if we would like to match the term *Tin oxide*, we would need a japelate for $n = 2$. Such a japelate can look as follows:

```
Rule: $0$
(
      {Token.string==~"$3$($4$)", Token.category!=IN}
      {Token.string==~"$5$($6$)", Token.category!=IN}
):binding
```

Clearly, these two japelates share a large proportion of code and will need to be updated in case of a change within the rule head. This problem has been addressed in the Java programming language by introducing the possibility of methods accepting a variable number of arguments of the same type. We transfer this notion to japelates as follows:

```
Rule: $0$
(
      ${Token.string==~"$3$($4$)", Token.category!=IN}$
):binding
```

This version of the `nonINMatcher` japelate accepts a variable number of arguments and repeats the expression enclosed by ${ and }$ while using a different value for parameter 4 in each line. Within NanOn, this construct has been excessively used to match annotations against ontology classes whose names consist of more than one word.

3.3 Abstract Instantiations

The introduction of parameters and variable number of arguments help to reduce the amount of JAPE code within a rule base. However, it should be observed that there is a trade-off between the redundancy within japelates and the redundancy within the actual rules: by introducing an additional parameter, we might be able to reduce the number of japelates and the redundancy within those, but we also increase the number of arguments that need to be passed within each japelate instantiation. For instance, we could generalize two japelates that are identical apart from one single expression (A and B in Fig.3) into a single japelate by introducing a parameter within this expression (expression C in Fig.3).

```
${Token.string==~"$3$($4$)", Token.category==~NN}$          (A)

${Token.string==~"$3$($4$)", Token.category==~JJ}$          (B)

${Token.string==~"$3$($5$)", Token.category==~$4$}$         (C)
```

Fig. 3. Expressions accepting sequences of strings with certain POS tags

If we have a considerable number of rules that have the same value for parameter 5, the overall redundancy within the rule base will increase. We now have to provide an additional value for every instantiation of the japelate in question. Moreover, sometimes it is convenient to be able to update certain values for parameter 5 all at once, e.g. in order to change the value *JJ* to *JJS* in all rules. In order to overcome the problem of redundancy within japelate instantiations and to provide more flexibility for reusing rule definitions, Semano introduces *abstract instantiations* — instantiations that do not bind all parameters to certain values and in turn need to be instantiated before being used for document annotation. Fig.4 shows an abstract instantiation `seqWithPOS` that includes expression C from Fig.3.

Overall, the Semano representation of the NanOn annotation rules is by an order of magnitude smaller than its JAPE representation due to the reduced amount of redundancy. It should be noted that the above difference is uninfluenced by the fact that the JAPE grammar provides the construct *MACRO* which can be applied to reuse identical parts within rule bodies. For instance, if the expression in line 7, Fig. 1 occurs several times within the rule body, we can introduce a MACRO and use it instead of the above expression. While this construct can be useful in large rule bodies, within the NanOn rule base we did not encounter a single rule body with two or more identical parts.

4 Extendability of Annotation Features

To maximize the recall and precision achieved by an ontology-based annotator, it is usually necessary to optimize the rule base for a particular corpus, ontology and the

```
1  JAPELATE HEADER:
2  0: LITERAL, Rule ID
3  1: LITERAL, Ontology IRI
4  2: ENTITY,  Class IRI
5  3: LITERAL, Case insensitivity expression (?i) or empty
6  4: ENTITY,  Main expression
7
8
9  ABSTRACT JAPELATE BODY:
10 $0$:  seqWithPOS($1$,$2$,$3$,NN,$4$)
```

Fig. 4. Abstract japelate instantiation `seqWithPOS` with a concrete value for parameter 4

particular application requirements in question. Consequently, the concrete characteristics of the generated annotations can vary significantly from application to application. Semano is a generic framework and accounts for the diversity of application requirements. In particular, it does not make any assumptions about application-specific annotation features, which can be used in japelates in the same way as the default features. In this section, we discuss how japelates can be extended to implement application-specific annotation rules.

Semano provides a selection of generic default japelates that can serve as a starting point for developing a rule base and can be extended into application-specific ones. In our example, we are using the default japelate `domainRangeMatcher` shown in Fig. 5. This japelate finds sentences enclosing an occurrence of a domain and a range class for a particular ontology relation.

Fig. 5 shows how a selection of default features are set in annotations of type *Mention* (lines 19-26). In addition to the ontology IRI (line 19) and the IRI of the ontology relation (line 23), which are set from the parameters 1 and 2, the japelate sets the references to the two entities connected to each other by this relation (lines 21-22). It also sets some meta information that is interpreted by the Semano annotation viewer presented in Section 6. For instance, it records that this annotation has been created by Semano (as opposed to annotations created manually by an engineer) and which rule and japelate have been used.

We now demonstrate how we can add a confidence value to annotations by extending `domainRangeMatcher`. In NanOn, we found it useful to be able to assign confidence values to annotation rules. Along other things, this enabled us to divide the entire rule base into *precise* and *recall-boosting* annotation rules. The former aim at identifying only candidates with a high chance of being correct and tend to be very specific. In contrast, the latter rules aim at identifying as many potential candidates as possible and are deliberately vague. We found recall-boosting rules to be very helpful at an early stage of rule base development to identify candidates for precise annotation rules. In fact, many precise annotation rules within the NanOn rule base that aim at identifying ontology relations are concretizations of recall-boosting rules.

One way to add a confidence feature to an annotation would be to set a particular value within the japelate, for instance, by adding the following line after line 26, Fig. 5:

```
1  JAPELATE PARAMETERS:
2  0: LITERAL, Rule ID
3  1: LITERAL, Ontology IRI
4  2: ENTITY,  Relation IRI
5  3: ENTITY,  Domain class IRI
6  4: ENTITY,  Range class IRI
7
8  JAPELATE BODY:
9  Rule: $0$
10 ({
11       Sentence contains {Mention.class=="$3$"},
12       Sentence contains {Mention.class=="$4$"}
13 }):binding
14 -->
15 {
16       ... Java code to initialize some annotation features ...
17
18       gate.FeatureMap features = Factory.newFeatureMap();
19       features.put("ontology", "$1$");
20       features.put("autoannotation", "true");
21       features.put("domain", domainAnnotation.getId());
22       features.put("range", rangeAnnotation.getId());
23       features.put("relation","$2$");
24       features.put("language", "en");
25       features.put("rule", "$0$");
26       features.put("japelate", "domainRangeMatcher");
27
28       ...  Java code creating the annotation ...
29 }
```

Fig. 5. Japelate `domainRangeMatcher` with confidence values

```
features.put("confidence","0.2");
```

This would be sufficient if all rules instantiating this japelate have the same confidence. However, later, we might notice that the confidence varies depending on the ontology relation in question. For instance, this japelate might yield good results for `materialPropertyOf`, but rather poor results for `hasFabricationProcess` as the latter relation has the same domain and range as another relation `inputMaterialOf`. To make it possible to set the confidence value in a rule rather than a japelate, we can add a parameter for confidence value after line 6, Fig. 5:

```
5: LITERAL, Confidence value
```

and then set the confidence of the annotation to parameter 5 as follows after line 26, Fig. 5:

```
features.put("confidence", "$5$");
```

As we expect this japelate to serve as a basis for recall-boosting rules rather then precise ones, we might find it cumbersome to repeat a low confidence value in numerous rules. In that case, we can add an abstract instantiation, e.g., `domainRangeMatcherLowConf`, which fixes confidence to some low value as shown in Fig. 6.

```
1 JAPELATE HEADER:
2 0: LITERAL, Rule ID
3 1: LITERAL, Ontology IRI
4 2: ENTITY, Relation IRI
5 3: ENTITY, Domain class IRI
6 4: ENTITY, Range class IRI
7
8 ABSTRACT JAPELATE BODY:
9 $0$: domainRangeMatcher($1$,$2$,$3$,$4$,0.2)
```

Fig. 6. Abstract japelate instantiation `domainRangeMatcherLowConf`

Thus, we can use the more flexible japelate `domainRangeMatcher` for rules with a low confidence value, while not having to repeat frequently occurring confidence values in other rules.

5 Semano in Action

We used Semano to annotate all volumes (1-12) of the *Journal of Applied Physics* from 2009 with the NanOn rule base. In total, the corpus consists of 3,358 articles. We pre-annotated the corpus with the standard GATE application *ANNIE*. On average, after this step each article contained around 8,000 annotations. Subsequently, we annotated the corpus with the NanOn rule base, which included 17,292 rules. On average, each rule within the rule base had 7 arguments.

The default workflow of corpus annotation is shown in Fig. 7. First, rules and japelates are compiled into JAPE files. Then, a set of transducers is initialized with a classified OWL ontology and the JAPE files produced in the previous step. Subsequently, they are run on the document corpus. During the initialization of transducers, GATE compiles JAPE scripts into binary Java code, which can take some time for a large rule base. Annotated documents are saved as XML files in GATE format and can be viewed and evaluated with the Semano annotation viewer as described in Section 6.

The annotation has been carried out on a MacBook Pro notebook with 16 GB of memory and 2.3 GHz Intel Core processor and yielded around 1000 ontology annotations per document, which corresponds to one ontology annotation for every four words within a document. The compilation of japelates and rules into JAPE files took in total 24 seconds for the entire rule base. The initialisation of transducers took in total 215 seconds. Thus, the additional time required for compiling japelates and rules into JAPE files is small in comparison to the overall initialization time. In our evaluation, we used a parallelized version of Semano, which has internally initialized four JAPE transducers. The annotation of the corpus with this parallelized version took on average 19 seconds

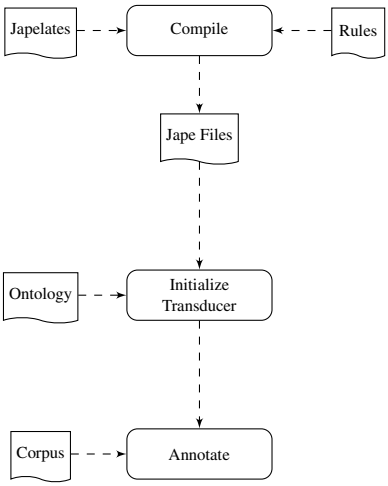

Fig. 7. Annotating a corpus with Semano

per document, which was 2.8 times faster than a single JAPE transducer that has been initialized in a usual way through GATE (54 seconds). We conclude that, on a machine with multiple cores, it is worthwhile parallelizing the annotation with Semano in order to speed up the annotation of large corpora.

6 GUI Components

In this section, we give an overview of the two GUI components within Semano — the annotation viewer and the rule base editor.

6.1 Annotation Viewer

Evaluating generated annotations is a crucial task in IE development. Appropriate software support is essential as it can take up a significant proportion of the overall project budget. Semano includes an annotation viewer that has been developed to make this process as efficient as possible. Fig.8 shows its general User Interface.

Supporting the evaluation of generated annotations is the key purpose of the annotation viewer. Given a GATE document, an ontology and a rule base, Semano annotation viewer highlights the relevant annotations within the current document and enables the engineer to record his/her feedback about their correctness. It provides a flexible way to filter the highlighted annotations within the document. For instance, it is possible to de-highlight annotations for particular japelates, rules and ontology entities.

Often it is during the evaluation of annotations that errors in rules or examples for further annotation rules are encountered. To support an efficient update of the rule base, the annotation viewer enables editing rules used within generated annotations or adding a new rule to the rule base and applying it to the current document. In this way, the new

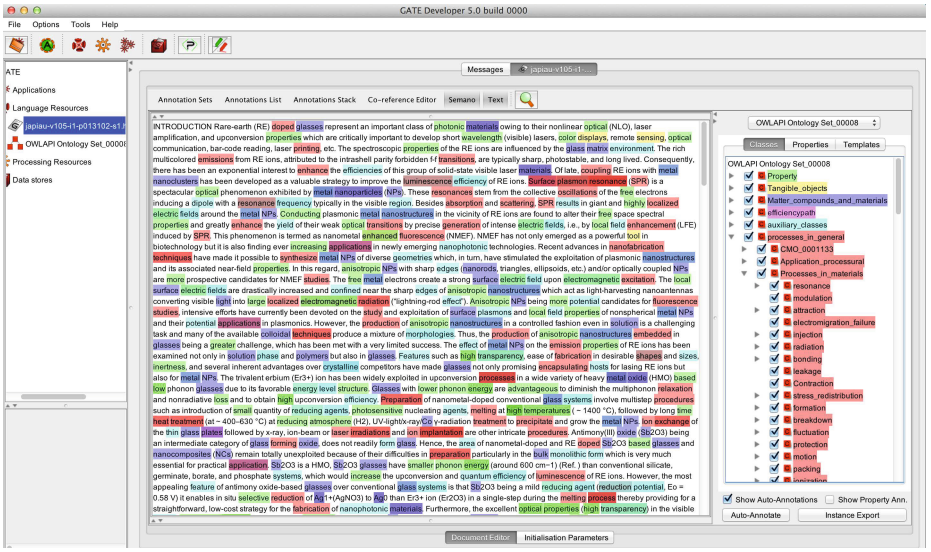

Fig. 8. Semano Annotation Viewer

rule can be tested immediately after its creation. A rule-wise document annotation is also significantly faster than the annotation with the entire rule base.

In addition to the above features, the annotation viewer provides a shortcut for annotating the current document with the entire rule base and ontology and exporting the annotations as ontology entities.

6.2 Rule Base Editor

In case of large rule bases, finding a particular rule becomes difficult. The Semano rule base editor provides flexible filters that enable an engineer to explore a particular part of the rule base. Further, it provides a convenient way of updating, adding and deleting rules from the rule base. Among other things, it assists the user in instantiating a particular japelate, for instance, by auto-completing the entity names for parameters of type ENTITY based on the available entities within the ontology.

7 Summary and Future Work

In this paper, we have discussed the role of ontologies within IE systems and presented Semano, which is a generic framework for ontology-based annotation. We have discussed various features of Semano such as its module for accessing OWL 2 DL ontologies within the GATE framework and access to services of various reasoners. We have presented Semano's rule base model and have discussed its role in modular rule base development. We have also discussed the generality of Semano and have shown an example of how this model can be used to implement rules given particular requirements.

We have further outlined the features of the two GUI components — a rule base editor and an annotation viewer — that provide a convenient access to a Semano rule base and Semano-generated annotations within documents, respectively. Overall, we can conclude that various features of Semano, in particular the modular rule base model, can help to significantly reduce the budget required for developing ontology-based annotators. In case of our example rule base NanOn, we have observed a significant increase in modularity: NanOn was by an order of magnitude smaller and significantly easier to update than its representation as a JAPE rule base, which is the original GATE rule base model.

As for any software framework, there are numerous features that could be added to Semano. One such feature is a more flexible type system for japelate parameters, which could be used to assist the engineer in creating japelate instantiations. Another direction for extending the functionality of Semano is to enable import of declarative ontology lexica such as lemon lexica [11].

References

1. Armas Romero, A., Cuenca Grau, B., Horrocks, I.: MORe: Modular combination of OWL reasoners for ontology classification. In: Cudré-Mauroux, P., et al. (eds.) ISWC 2012, Part I. LNCS, vol. 7649, pp. 1–16. Springer, Heidelberg (2012)
2. Bail, S., Glimm, B., Gonçalves, R.S., Jiménez-Ruiz, E., Kazakov, Y., Matentzoglu, N., Parsia, B. (eds.): Informal Proceedings of the 2nd International Workshop on OWL Reasoner Evaluation (ORE 2013), Ulm, Germany, July 22. CEUR Workshop Proceedings, vol. 1015. CEUR-WS.org (2013)
3. Bishop, B., Kiryakov, A., Ognyanoff, D., Peikov, I., Tashev, Z., Velkov, R.: Owlim: A family of scalable semantic repositories. Semant. Web 2(1), 33–42 (2011), http://dl.acm.org/citation.cfm?id=2019470.2019472
4. Bizer, C., Lehmann, J., Kobilarov, G., Auer, S., Becker, C., Cyganiak, R., Hellmann, S.: Dbpedia - a crystallization point for the web of data. Web Semant. 7(3), 154–165 (2009), http://dx.doi.org/10.1016/j.websem.2009.07.002
5. Bollacker, K., Evans, C., Paritosh, P., Sturge, T., Taylor, J.: Freebase: A collaboratively created graph database for structuring human knowledge. In: Proceedings of the 2008 ACM SIGMOD International Conference on Management of Data, SIGMOD 2008, pp. 1247–1250. ACM, New York (2008), http://doi.acm.org/10.1145/1376616.1376746
6. Cunningham, H., Maynard, D., Bontcheva, K., Tablan, V., Aswani, N., Roberts, I., Gorrell, G., Funk, A., Roberts, A., Damljanovic, D., Heitz, T., Greenwood, M.A., Saggion, H., Petrak, J., Li, Y., Peters, W.: Text Processing with GATE, Version 6 (2011), http://tinyurl.com/gatebook
7. Horridge, M., Bechhofer, S.: The owl api: A java api for working with owl 2 ontologies. In: Hoekstra, R., Patel-Schneider, P.F. (eds.) OWLED. CEUR Workshop Proceedings, vol. 529. CEUR-WS.org (2009)
8. Kazakov, Y., Krötzsch, M., Simančík, F.: ELK: a reasoner for OWL EL ontologies. System description, University of Oxford (2012), http://code.google.com/p/elk-reasoner/wiki/Publications
9. Krötzsch, M., Vrandecic, D.: Semantic mediawiki. In: Foundations for the Web of Information and Services, pp. 311–326. Springer (2011)

10. Lawley, M.J., Bousquet, C.: Fast classification in Protégé: Snorocket as an OWL 2 EL reasoner. In: Meyer, T., Orgun, M.A., Taylor, K. (eds.) Australasian Ontology Workshop 2010 (AOW 2010): Advances in Ontologies. CRPIT, vol. 122, pp. 45–50. ACS, Adelaide (2010); Winner of Payne-Scott Best Paper Award
11. McCrae, J., Spohr, D., Cimiano, P.: Linking lexical resources and ontologies on the semantic web with lemon. In: Antoniou, G., Grobelnik, M., Simperl, E., Parsia, B., Plexousakis, D., De Leenheer, P., Pan, J. (eds.) ESWC 2011, Part I. LNCS, vol. 6643, pp. 245–259. Springer, Heidelberg (2011)
12. Mika, P., Potter, T.: Metadata statistics for a large web corpus. In: Bizer, C., Heath, T., Berners-Lee, T., Hausenblas, M. (eds.) LDOW. CEUR Workshop Proceedings, vol. 937. CEUR-WS.org (2012)
13. OWL Working Group, W.: OWL 2 Web Ontology Language: Document Overview. W3C Recommendation (October 27, 2009), http://www.w3.org/TR/owl2-overview/
14. Shearer, R., Motik, B., Horrocks, I.: HermiT: A Highly-Efficient OWL Reasoner. In: Proceedings of the 5th International Workshop on OWL: Experiences and Directions, OWLED 2008 (2008)
15. Sirin, E., Parsia, B., Cuenca Grau, B., Kalyanpur, A., Katz, Y.: Pellet: A practical OWL-DL reasoner. Web Semantics: Science, Services and Agents on the World Wide Web 5(2), 51–53 (2007), http://dx.doi.org/10.1016/j.websem.2007.03.004
16. Spackman, K.A., Campbell, K.E., Cote, R.A.: SNOMED RT: A reference terminology for health care. In: Proceedings of the AIMA Fall Symposium, pp. 640–644 (1997)
17. Suchanek, F.M., Kasneci, G., Weikum, G.: YAGO: A Large Ontology from Wikipedia and WordNet. Web Semantics: Science, Services and Agents on the World Wide Web 6, 203–217 (2008)
18. Tipney, H., Hunter, L.: Knowledge-Driven Approaches to Genome-Scale Analysis. John Wiley & Sons, Ltd. (2010), http://onlinelibrary.wiley.com/doi/10.1002/9780470669716.ch2/summary
19. Tsarkov, D., Horrocks, I.: FaCT++ Description Logic Reasoner: System Description. In: Furbach, U., Shankar, N. (eds.) IJCAR 2006. LNCS (LNAI), vol. 4130, pp. 292–297. Springer, Heidelberg (2006)
20. Wheeler, D.L., Chappey, C., Lash, A.E., Leipe, D.D., Madden, T.L., Schuler, G.D., Tatusova, T.A., Rapp, B.A.: Database resources of the National Center for Biotechnology Information. Nucleic Acids Research 28(1), 10–14 (2000), http://dx.doi.org/10.1093/nar/28.1.10