A Ontea – Pattern based Semantic Annotation Platform

A.1 Basic Information

Ontea tool analyze document or text using a regular expression patterns and detects equivalent semantics elements according to defined domain ontology. Several cross application patterns are defined but to achieve good results new patterns need to be defined for each application. Ontea also creates new ontology individual of defined class and assignees detected ontology elements/individuals as properties of defined ontology class.

The method used in Ontea is comparable particularly with methods sucha s C-PANKOW, KIM, or SemTag. It works over text applicable to an application domain that is described by a domain ontological model and uses regular expressions to identify relations between text and a semantic model. In addition to having pattern implementation over regular expressions, created Ontea's architecture allows simply implementation of other methods based on patterns such as wrapers, solutions using document structure, language patterns, e.g. C-PANKOW and many others.

A.1.1 Basic Terms

Semantic Annotation of Text	Identification of formalized objects in texts
Ontology	Formalized model of problem environment understandable by computer system
Regular expression patterns	regular expression is a string that is used to describe or match a set of strings, according to certain syntax rules

A.1.2 Method Description

Ontea works on text, in particular domain described by domain ontology and uses regular expression patterns for semi-automatic semantic annotation. In Ontea we try to detect ontology elements within the existing application/domain ontology model. It means that with the Ontea annotation engine we want to achieve the following objectives:

- Detect meta data from the text
- Prepare improved structured data for later computer processing
- Structured data are based on the existing application ontology model

The tool results can be made more precise by connecting Ontea with other tools for lemmatization (e.g. The Morphonary[1] tool for lemmatization of Slovak) and also by estimating relevance of new created individuals by information retrieval tools such a s RFTS or Lucene.

A.1.3 Scenarios of Use

Ontea can be executed in 3 different scenarios:

- *Ontea*: searching relevant concepts in knowledge base (KB) according to generic patterns
- *Ontea creation*: creating new individuals of concrete application specific objects found in text
- *Ontea IR*: Similar as previous with the feedback of information retrieval tool (e.g. Lucene) to get relevance computed above word occurrence.

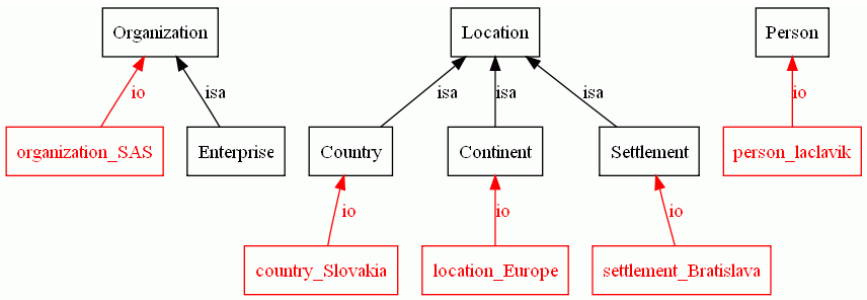


Figure 1 Simple ontology used in examples

The following texts, one in Slovak and one in Englis, in Table 1 and ontology with several instances (Figure 1) illustrate the Ontea method. Moreover the table shows examples of used pattersn based on regular expressions. For English only one regular expression was used to find one or two words beginning with a capital letter.

Table 1 Example Text

Príklad	Text	Vzory – regulárne výrazy
1	Michal Laclavik works for Slovak Academy of Sciences located in Bratislava, the capital of Slovakia	\\b(\\p{Lu}[a-z]+ +\\p{Lu}[a-z]+|\\p{Lu}[a-z]+)\\b
2	Automobilka KIA sa rozhodla investovať pri Žiline, kde vybudovala svoju prvú továreň v Európe. Kontakt: Kia Motors Slovakia, s.r.o. P.O.Box 2 01301 Teplička nad Váhom Slovakia	\\b([\\p{Lu}][-&\\p{L}]+[ ]*[-&\\p{L}]*[ ]*[-&\\p{L}]*)[,]+s\\.r\\.o\\. *Organization* \\b[0-9]{3}[ ]*[0-9]{2} +(\\p{Lu}[^\\s,.]+[ ]*[^0-9\\s,.]*[ ]*[^0-9\\s,.]*)[ ]*[0-9\\n,]+ *Settlement* (v|pri) +(\\p{Lu}\\p{L}+) *Location*
3	Car maker KIA Motors decided to build new plant in Slovakia near town of	(in|near) +(\\p{Lu}\\p{L}+) *Location*

[1] http://nazou.fiit.stuba.sk/home/?page=morphonary

	Zilina. It is its first plant in Europe. Contact: Kia Motors Slovakia, ltd. P.O.Box 2 01301 Teplicka nad Vahom Slovakia	(city|town) of (\\p{Lu}\\p{L}+ *\\p{Lu}}\\p{L}*) *Settlement* \\b([\\p{Lu}][-&\\p{L}]+[ ]*[-&\\p{L}]*[ ]*[-&\\p{L}]*)[,]+ (Inc|Ltd)[.\\s]+ *Organization*

In the English text example showed only finding of instances present in a knowledge base, see Figure 1 and Table 2, 1st row. The Slovak text example demonstrated not only searching for instances within a knowledge base but also their creation. The example uses three patterns for Slovak:

- Searching for a company by „s.r.o.“ (public limited company)
- Searching for residence/domicile by ZIP code
- Searching for geographical location by means of prepositions *in*, *near* followed by a word beginning with a capital letter.

Table 2, 2nd and 3rd rows give the results. Instances created from the 2nd row are shown also on Figure 2.

Table 2 Annotation result

Example	Method	Annotation Results
1	Ontea	Michal Laclavik *Person* Slovak Academy *Organization* Bratislava *Settlement* Slovakia *Country*
2	Ontea create	Žiline *Location* Európe *Location* Kia Motors Slovakia *Organization* Teplička nad Váhom *Settlement*
2	Ontea create, lematizácia	Žilina *Location* Európa *Continent* Kia Motors Slovakia *Organization* Teplička nad Váh *Settlement*
3	Ontea create	Slovakia *Country* Zilina *Settlement* Europe *Continent* Kia Motors Slovakia *Organization*

To produce results in row 3 in Table 2 we used lemmatization of found texts that enables more precise creation of instances in the nominative in cases „Žilina“ and „Európa“, however, location „Teplička nad Váhom“ came up wrong, as „Teplička nad Váh“. It would be appropriate to use lemmatization only for several patterns, not for all of them. Furthermore, we can direct our attention to „Európa“ that is not of a type *Location* but *Continent* because algorithm found it in a knowledge base using inference since *Continent* is a subclass of *Location*. In a creation process, algorithm first looks into the knowledge base to find out whether instance of such type or inferred instance already exists.

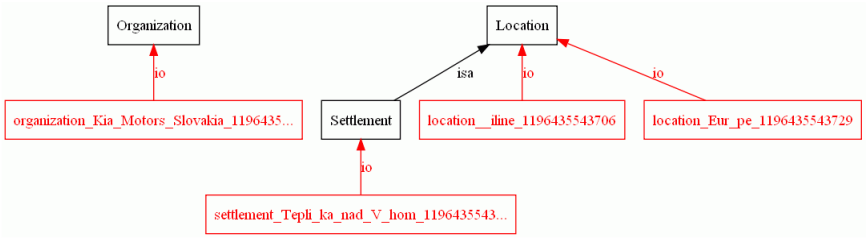


Figure 2 created instances in Slovak text – Ontea create.

Use of Ontea, 1st scenario is also described on example from Job Offer Application.

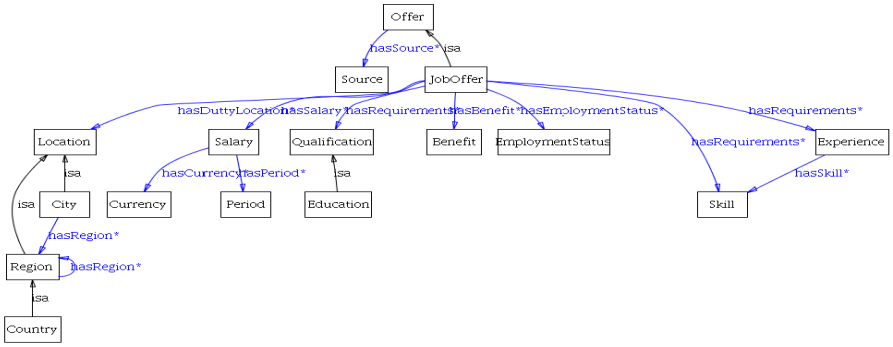
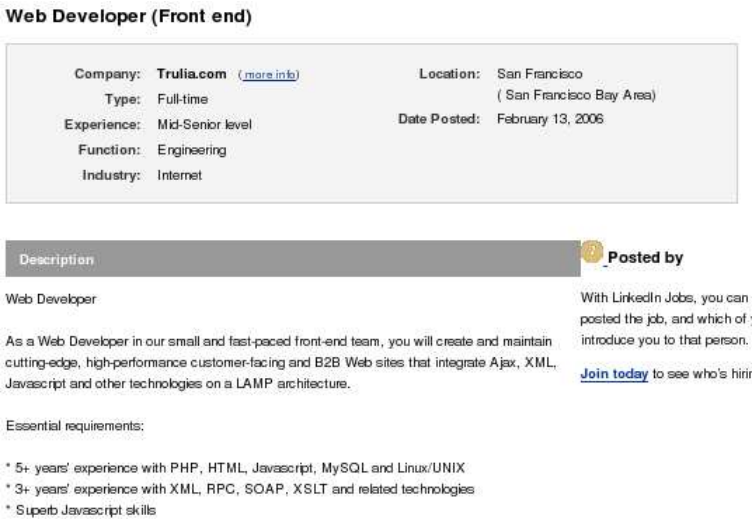


Figure 3 Job Offer Ontology

On Figure 3 main components of Job Offer ontology are displayed. Important fragments on ontology are Location or Skills individuals which can be then detected by annotation.



Web Developer (Front end)

Company: **Trulia.com** (more info)
Type: Full-time
Experience: Mid-Senior level
Function: Engineering
Industry: Internet

Location: San Francisco
(San Francisco Bay Area)
Date Posted: February 13, 2006

Description

Posted by

Web Developer

As a Web Developer in our small and fast-paced front-end team, you will create and maintain cutting-edge, high-performance customer-facing and B2B Web sites that integrate Ajax, XML, Javascript and other technologies on a LAMP architecture.

With LinkedIn Jobs, you can posted the job, and which of introduce you to that person.

Join today to see who's hirir

Essential requirements:

* 5+ years' experience with PHP, HTML, Javascript, MySQL and Linux/UNIX
* 3+ years' experience with XML, RPC, SOAP, XSLT and related technologies
* Superb Javascript skills

Figure 4 Web Document of Job Offer

On Figure 5 the individual of the Job Offer is created based on the semantic annotation of a Job Offer document (Figure 3), using simple regular expression patterns (see regular expression patterns chapter) where main individuals can be detected by the title property such as skillXML or skillPHP individuals. Other specialized patterns such as pattFullTime are used to detect concrete job offer properties – jtPermanent individual,

which represents a permanent job position. In this example the job offer location – San Francisco id identified by a regular expression `([-A-Za-z0-9]+ [ ]+[-A-Za-z0-9]+)`, because individual locSF has the property title „San Francisco“. Similarly, other ontology elements are detected. Some regular expressions search for ontology individuals, other ontology classes and others such as pattFullTime annotate a job offer by a concrete individual jtPermanent if expression `(Full[ -]Time)` is found. Systems detect ontology elements based on domain ontology. In this example it is ontology of job offers.

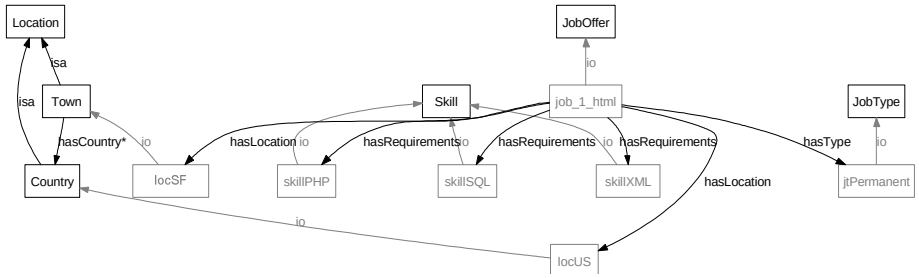


Figure 5 Offer Individual Created by Ontea

Detected ontology individuals are then assigned as properties of job offer, thus ontology instance of job offer is created out of its text representation in the NAZOU pilot application.

A.1.4 External Links and Publications

- Ontology based Text Annotation – OnTeA; Michal Laclavik, Martin Seleng, Emil Gatial, Zoltan Balogh, Ladislav Hluchy; Information Modelling and Knowledge Bases XVIII. IOS Press, Amsterdam, Marie Duzi at al., Frontiers in AI, Vol. 154, February 2007, pp.311-315. ISBN 978-1-58603-710-9, ISSN 0922-6389.
- Ontea: Pattern based Semantic Annotation Platform, SourceForge.net project, http://ontea.sourceforge.net/
- Hatcher E., Gospodnetić O., Lucene in Action, Manning (12 Jan 2005), ISBN: 1932394281
- Log4J, Java-based logging utility, Apache Software Foundation. (http://logging.apache.org/log4j)
- Jena, http://jena.sf.net/
- Sesame, http://openrdf.org/

A.2 Integration Manual

Ontea is developed in Java (Standard Edition 5) and distributed as a `jar` archive. Access to the functionality of the tool is provided through Java Interface. Ontea is not a stand-alone application; the tool is proposed to be included in other application/tool, which will call the Ontea methods, however several classes are implemented with *main()* method to run some test and example functionality.

A.2.1 Dependencies

Ontea uses following libraries:

- Log4J logging utility
- Jena or Sesame semantic web library depending on used version
- Lucene information retrieval library in case of use in IR scenario
- Morphonary tool in case of connection with lemmatizator

A.2.2 Installation

Deploying Ontea into other application requires the following steps (Java Integrated Development Environment and Apache Ant should be used):

1. download all Ontea files from ontea.sf.net
2. `ant start` to start the demo or `ant dist` to create jar file and use library

You can run and see ontea.example.text.Annotation class for more details. Directory "example" in the ontea root directory contains configuration files, text to be annotated and simple ontology presented in the scenario.

A.2.3 Configuration

Ontea use property file which need to be defined while creating onto.sesame.`Memory` instance. See Java Doc for more details.

Values such as Sesame repository type, location or ontology namespaces need to be defined. See `onto.sesame.Config` Java Doc for more details.

`ontea.config` package contain of `pattern.property` files for different languages or applications where regular expression patterns can be modified.

A.2.4 User Guide

Basic Use

If you want to use Ontea without deep knowledge of its structure and functionalities, you should build your code based on examples in `ontea.example.text` package. Class `Annotation` shows example how to annotate text from "example" directory by applying regular expression patterns from user defined property file (also in example directory).

Annotation examples are also provided within JUnit test classes e.g. `ontea.core.test.TestPatternRegExp` class, showing annotation of String using defined regular expression.

Code Structure

Main Ontea classes and interfaces are located in `ontea.core` package. Main objects are:

- Pattern
- Result

Ontea annotation is build on pattern based approach, thus `Pattern` interface is one of main Ontea objects. Currently only one implementation of `Pattern` is provided:

`PatternRegExp`, which annotates using regular expression patterns. Each `Pattern` implementation has to implement annotates method, which annotates given text and returns Set of `Results`. Result represents result of annotation, which is in fact individual of certain type or class. So far there are two `Result` super classes: `ResultRegExp` and `ResultOnto`. While `Result` is general individual not depending of used pattern or ontology/model implementation, `Result` extensions are created or found by applying result transformers on `Result` instances or `Result` sets.

These transformers are located in `ontea.transform` package. Package contain `ResultTransformer` interface with methods for transforming `Result` and `Result` set and also its implementations. Some of implementations are annotation result lemmatization, ontology knowledge base mapping or relevance identification using information retrieval methods.

Other packages and classes are related to transformers implementation, e.g manipulating sesame repository or integrating lucene indexing functionality to identify relevance.

Use of transformers can be seen in `ontea.example.text.Annotation` class, which shows use of sesame transformer as well as lemmatization on Slovak text.

A.3 Developer Manual

A.3.1 Tool Structure

Architecture of the system contains similar elements as the main annotation algorithm described in next chapter. Inputs are text resources (HTML, email, plain text) which need to be annotated as well as corresponding patterns from property files. An output is a new ontology individual, which corresponds to the annotated text. Properties of this individual are filled with detected ontology individuals according to defined patterns.

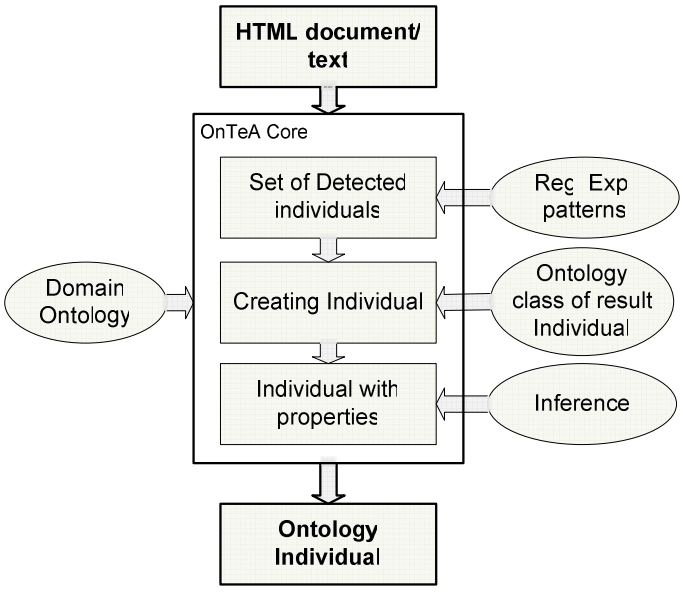


Figure 6 Ontea architecture

When extending the code of Ontea some of following classes and interfaces need to be extended and implemented:

- ontea.core.Pattern: interface to adopt different pattern annotation techniques
- onetea.core.Result: class representing results of pattern annotation – instances of defined type.
- ontea.transform.ResultTransformer: interface transforming results of annotation to different type or quality of results e.g. concrete ontology mapping, knowledge base implementation or result quality checking.

Ontea works with RDF/OWL Ontologies. It is implemented in Java using Jena Semantic Web Library or Sesame library. In both implementation inference is used to achieve better results.

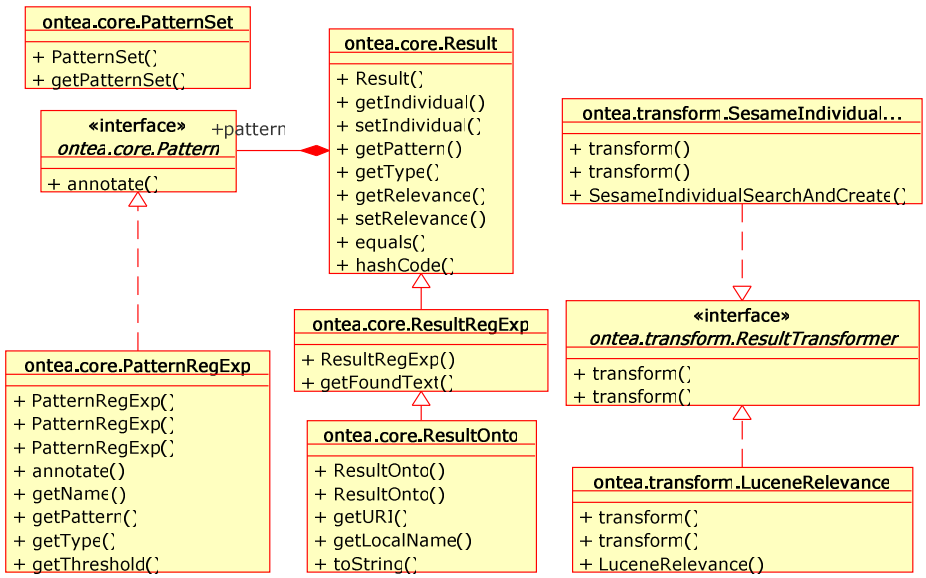


Figure 7 Basic classes of Ontea platform.

Ontea tool can be easily integrated with external tools. Some tools can be integrated by implementation of result transformers and other need to be integrated directly.

- *MapReduce*: Large scale semantic annotation using MapReduce Architecture – is main topic of this article. Integration with Hadoop requires implementation of Map and Reduce methods as described in next chapter.
- *Language Identification*: In order to use correct regexes or other patterns, often we need to identify language of use. For this reason it is convenient to integrate Ontea with language detection tool. We have tested Ontea with Nalit. Nalit is able to identify Slovak and English texts as well as others if trained.

As already mentioned some integration can be done by implementing Result transformers:

- *Lemmatization:* When concrete text is extracted as representation of an individual, often we need to lemmatize found text to found or create correct instance. For example capital of Slovakia can be identified in different morphological forms: *Bratislava*, *Bratislave*, *Bratislavu*, or *Bratislovou* and by lemmatization we can identify it always as individual *Bratislava*. We have tested Ontea with Slovak lemmatizer Morphonary. It is also possible to use

lemmatizers or stemmers from Snowball project[2], where java code can be generated.

- *Relevance Identification*: When new instance is being created or found, it is important to decide on instance relevance. This can be solved using information retrieval methods and tools such as Lucene[3]. When connecting with Lucene, Ontea asks for percentage of occurrence of matched regular expression pattern to detected element represented by word on used document set. Document set need to be indexed by Lucene. Example can be *Google, Inc.* matched by pattern for company search: *\\s+([-A-Za-z0-9][ ]*[A-Za-z0-9]*),[ ]*Inc[.\\s]+*", where relevance is computed as "Google, Inc." occurrence divided by "Google" occurrence. Use of Lucene is related to *Ontea IR* scenario and *LuceneRelevance* implementation of *ResultTransformer* interface. Similarly, other relevance algorithms such as cosine measure can be implemented. This was used for example in SemTag.
- *OWL Instance Transformation*: Sesame, Jena: Transformation of found key – value pairs into RDFS or OWL instances in Sesame or Jena API. With this integration, Ontea is able to find existing instances in knowledge base if existing and create new once if no instance found in DB. Ontea also use inference to found appropriate instance. For example if Ontea process sentence "Slovakia is in Europe." using pattern for location detection *(in|near) +(\\p{Lu}\\p{L}+)* following type value pair is detected *Location: Europe*. If we have Location ontology with Subclasses as Continents, Settlements, Countries or Cities and Europe is already present as instance of continent, Ontea can detect existing Europe instance in knowledge base using inference.

A.3.2 Method Implementation

The underlying principle of the Ontea algorithm can be described by the following steps:

1. The text of a document is loaded.
2. The text is proceed by defined regular expressions and if they are found, corresponding ontology individual according to rest of pattern properties is added to a set of found ontology individuals.
3. If no individual was found for matched pattern and createInstance property is set, a simple individual of the class type contained in the hasClass property is created with only property rdf:label containing matched text.
4. Such process is repeated for all regular expressions and the result is a set of found individuals.
5. An empty individual of the class representing proceed text is created and all possible properties of such ontology class are detected from the class definition.
6. The detected individual is compared with the property type and if the property type is the same as the individual type (class), such individual is assigned as this property.
7. Such comparison is done for all properties of a new individual corresponding with the text/document as well as for all detected individuals.

[2] http://snowball.tartarus.org/

[3] http://lucene.apache.org/

The algorithm also uses inference in order enable assignment of a found individual to the corresponding property also if the inferred type of a found individual is the same as the property type. The weak point of the algorithm is that if the ontology definition corresponding with the detected text contains several properties of the same type, in this case detected individuals cannot be properly assigned. This problem can be overcome if algorithm is used only on creation of individuals of different property types. Crucial steps of the algorithms as well as inputs and outputs can be seen also on figure 6.

Regular Expression Patterns

Regular expression patterns are the key element of the Ontea algorithm. Usually for each problem domain we need to define new, problem specific patterns to match the ontology elements in the text. However several cross application patterns exist:

- Matching one word starting with capital letter: `([A-Z][-A-Za-z0-9]+)`
- Two words pattern: `([A-Z][A-Za-z0-9]+[\s]+[A-Z][A-Za-z0-9]+)`
- Similar for 3 and 4 words pattern.

When individuals in the reference domain ontology contain plain text labels describing or naming the individuals, they can be detected by Ontea using the patterns mentioned above. Even when using such simple patterns, we can achieve satisfactory results.. If the reference ontology contains critical amount of individuals with assigned text labels, results of annotation are satisfactory with above mentioned cross application patterns. A good example is the location ontology, which is used in both examples provided in this paper. The location ontology contains concepts such as regions, countries, settlements, mountains, rivers or lakes as well as individuals of such classes. It is possible to create such ontology with concrete individuals of towns, settlements, mountains or rivers. Such data are available on the internet[4]. We have also created such ontology containing all geographical data for Slovakia.

When keywords such as “Danube” or “Bratislava” appear in the text, correct individuals are detected by Ontea, where the Danube is an individual of a river and Bratislava is an individual of the Capital subclass of the settlement and the town. Similar detection can be achieved also with concepts such as a skill, a company or a category when ontology consists of a critical mass of individuals.

As already described, Ontea not only detects but also creates individuals if patterns are set up that way. A good example is again the location ontology. In many web pages for instance job offers, location is referenced by text “Location: City or Region Name”. When we convert web pages to a plain text, a regular expression pattern can be easily set up as follows: `Location:[\s]*([A-Z][-a-zA-Z]+[\s]*[A-Za-z0-9]*)`. This will match one or two words after the “Location:” string. If document contains e.g. “Location: New York” and “New York” text is not found in any individual in the reference ontology, we can create a new simple individual of the region type (this is set up in *Class* property of pattern) with *rdfs:label* “New York”. In the future if New York is found in another document, the same individual is detected. Note that if we would create e.g. “New York City” individual, one can be sure that New York City is not a region but rather it’s subclass - the town in our location ontology. If we would change

[4] GEOnet Names Server, Names, 2006, http://earth-info.nga.mil/gns/html/cntry_files.html

manually such individual to be an individual of the town class, it would be updated automatically in all detected data.

We think that with Ontea it is possible to detect or create ontology elements within the reference ontology with satisfactory results. This can be achieved automatically or semi automatically, when an expert can review and update the results.

Each pattern in Ontea contains of several properties which are defined in java property files:

- PATTERN: contain regular expression pattern which is search in text
- CLASS: represents URI of ontology class. Individuals are searched within this class.
- CREATE: If set to True, and PATTERN was found in text but not found in knowledge base, Ontea will create simple individual of CLASS type in knowledge base with only property of found text as rdfs:label.

See pattern files in example dir for examples of objects and patterns of Ontea.

A.4 Manual for Adaptation to Other Domains

The Ontea tool can be used in different application domains. Used annotation method is generic and it can work with any text and OWL based ontology model.

A.4.1 Configuring to Other Domain

When using Ontea in other domain it is necessary to provide following modifications:

- To change application or domain ontology
- To change or modify used regular expressions

The tool search or create ontology individuals in scope of domain ontology model, thus change of ontology model is related to Ontea core functionality.

When changing regular expression patterns, we can still use some generic patterns or reuse patterns for specific individuals/objects. See section on regular expression patterns.

If Ontea is used with Information Retrieval tools such as Lucene or RFTS, depending on language of processed texts, appropriate lemmatization tool need to be used.

A.4.2 Dependencies

Log4J is involved domain independently into the Ontea.

Lemmatization tool need to be used based on language of texts in the domain. E.g. *Morponary* for Slovak or *Porter's algorithm* for English.