



GEOLIFT - Spatial mapping framework for enriching RDF datasets with Geo-spatial information.

Abstract:

This manual presents the spatial mapping component dubbed GEOLIFT. The goal of GEOLIFT is to enrich RDF datasets with geo-spatial information. To achieve this goal, GeoLit relies on three atomic modules based on dereferencing, linking and NLP. GEOLIFT was implemented in Java, is open-source and can be accessed at <https://github.com/GeoKnow/GeoLift/>.

Contents

1	Introduction	4
2	Assumptions	4
3	Technical Approach	5
3.1	Architecture	5
3.2	Using Dereferencing	6
3.3	Using Linking	7
3.4	Using Named Entity Recognition	9
4	Developers' Manual	10
4.1	GEOLEFT packages	10
4.2	GEOLEFT Modules	12
4.2.1	Dereferencing Module	12
4.2.2	Linking Module	13
4.2.3	NLP Module	15
4.2.4	Conformation Module	15
4.2.5	Filter Module	19
4.3	GEOLEFT Operators	19
4.3.1	Split Operator	20
4.3.2	Merge Operator	20
5	GeoLift RDF Specification Paradigm	20
5.1	GEOLEFT Execution Workflow	21
5.1.1	RDF Specification Resources	21
6	Running GeoLift From Command-Line	24
7	Conclusions	24

1 Introduction

Manifold RDF data contain implicit references to geographic data. For example, music datasets such as *Jamendo* include references to locations of record labels, places where artists were born or have been, etc. The aim of the spatial mapping component, dubbed GEOLIFT, is to retrieve this information and make it explicit. In the following, we begin by presenting the basic assumptions that influence the development of the first component of GEOLIFT. Then, we present the technical approach behind GEOLIFT. Finally, we present the detailed developers' manual of GEOLIFT.

2 Assumptions

Geographical information can be mentioned in three different ways within Linked Data:

1. *Through dereferencing*: Several datasets contain links to datasets with explicit geographical information such as *DBpedia* or *LinkedGeoData*. For example, in a music dataset, one might find information such as `http://example.org/Leipzig`
`owl:sameAs`
`http://dbpedia.org/resource/Leipzig`.

We call this type of reference *explicit*. We can now use the semantics of RDF to fetch geographical information from DBpedia and attach it to the resource in the other ontology as `http://example.org/Leipzig` and `http://dbpedia.org/resource/Leipzig` refer to the same real-world object.

2. *Through linking*: It is known that the Web of Data contains an insufficient number of links. The latest approximations suggest that the Linked Open Data Cloud alone consists of 31+ billion triples but only contains approximately 0.5 billion links (i.e., less than 2% of the triples are links between knowledge bases). The second intuition behind our approach is thus to use link discovery to map resources in an input knowledge base to resources in a knowledge that contains explicit geographical information. For example, given a resource `http://example.org/Athen`, GEOLIFT should aim to find a resource such as `http://dbpedia.org/resource/Athen` to map it with. Once having established the link between the two resources, GEOLIFT can then resolve to the approach defined above.

3. *Through Natural Language Processing*: In some cases, the geographic information is hidden in the objects of data type properties. For example, some datasets contain biographies, textual abstracts describing resources, comments from users, etc. The idea here is to use this information by extracting Named Entities and keywords using automated Information Extraction techniques. Semantic Web Frameworks such as FOX¹ have the main advantage of providing URIs for the keywords and entities that they detect. These URIs can finally be linked with the resources to which the datatype properties were attached. Finally, the geographical information can be dereferenced and attached to the resources whose datatype properties were analyzed.

The idea behind GEOLIFT is to provide a generic architecture that contains means to exploit these three characteristics of Linked Data. In the following, we present the technical approach underlying GEOLIFT.

3 Technical Approach

3.1 Architecture

GEOLIFT was designed to be a modular tool which can be easily extended and re-purposed. In its first version, it provides two main types of artifacts:

1. *Modules*: These artifacts are in charge of generating geographical data based on RDF data. To this aim, they implement the three intuitions presented above. The input for such a module is an RDF dataset (in Java, a *Jena Model*). The output is also an RDF dataset enriched with geographical information (in Java, an enriched *Jena Model*). Formally, a module can thus be regarded as a function $\mu : \mathcal{R} \rightarrow \mathcal{R}$, where \mathcal{R} is the set of all RDF datasets.
2. *Operators*: The idea behind operators is to enable users to define a workflow for processing their input dataset. Thus, in case a user knows the type of enrichment that is to be carried out (using linking and then links for example), he can define the sequence of modules that must be used to process his dataset. Note that the format of the input and output of modules is identical. Thus, the user is empowered to create workflows of arbitrary complexity by simply connecting modules. Formally, an operator can be regarded as a function $\varphi : \mathcal{R} \cup \mathcal{R}^2 \rightarrow \mathcal{R} \cup \mathcal{R}^2$.

¹<http://fox.aksw.org>

The corresponding architecture is shown in Figure 1. The input layer allows reading RDF in different serializations. The enrichment modules are in the second layer and allow adding geographical information to RDF datasets by different means. The operators (which will be implemented in the future version of GEOLIFT) will combine the enrichment modules and allow defining a workflow for processing information. The output layer serializes the results in different format. The enrichment procedure will be monitored by implementing a controller, which will be added in the future version of GEOLIFT.

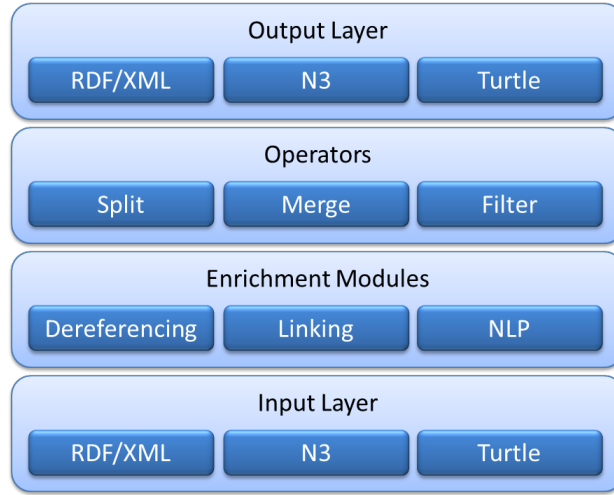


Figure 1: Architecture of GEOLIFT

In the following, we present the implementation of the three intuitions presented above in GEOLIFT.

3.2 Using Dereferencing

For datasets which contain `owl:sameAs` links, we dereference all links from the dataset to other datasets by using a content negotiation on HTTP as shown in Figure 2. This returns a set of triples that needs to be filtered for relevant geographical information. Here, we use a predefined list of attributes that links to geographical information. Amongst others, we look for `geo:lat`, `geo:long`, `geo:lat_long`, `geo:line` and `geo:polygon`. The list of retrieved property values can be configured.



Figure 2: Content Negotiation as used by GEOLIFT (courtesy of W3C)

3.3 Using Linking

As pointed out before, links to geographical resources do not occur in several knowledge bases. Here, we rely on the metrics implemented in the LIMES framework² [5, 4, 6] to link the resources in the input dataset with geographical datasets. LIMES, the **Link** Discovery Framework for **Metric** Spaces, is a framework for discovering links between entities contained in Linked Data sources. LIMES is a hybrid framework [4] that combines the mathematical characteristics of metric spaces as well prefix-, suffix- and position filtering to compute pessimistic approximations of the similarity of instances. These approximations are then used to filter out a large amount of those instance pairs that do not suffice the mapping conditions. By these means, LIMES can reduce the number of comparisons needed during the mapping process by several orders of magnitude and complexity without losing a single link. The architecture of LIMES is shown in Figure 3

Linking using LIMES [4, 3] can be achieved in three ways:

1. *Manually*, by the means of a link specification [4], which is an XML-description of (1) the resource in the input and target datasets that are to be linked and (2) of the similarity measure that is to be employed to link these datasets.
2. *Semi-automatically* based on active learning [7, 8, 9]. Here, the idea is that if the user is not an expert and thus unable to create a link specification, he can simply provide the framework with positive and negative examples iteratively. Based on these examples, LIMES can compute links for mapping resources with high accuracy.
3. *Automatically* based on unsupervised machine learning. Here, the user can simply specify the sets of resources that are to be linked with each other. LIMES implements both a deterministic and non-deterministic

²<http://limes.sf.net>

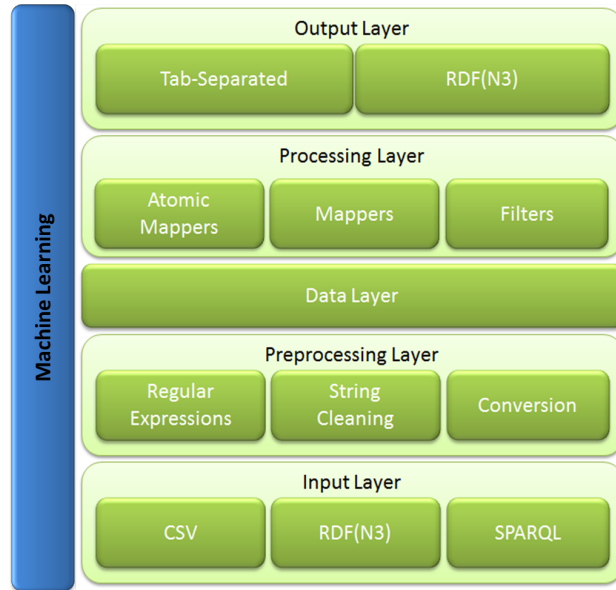


Figure 3: Architecture of LIMES

machine-learning approaches that optimize a pseudo-F-measure to create a one-to-one mapping.

The techniques implemented by LIMES can be accessed via the SAIM user interface³, of which a screenshot is shown in Figure 4.

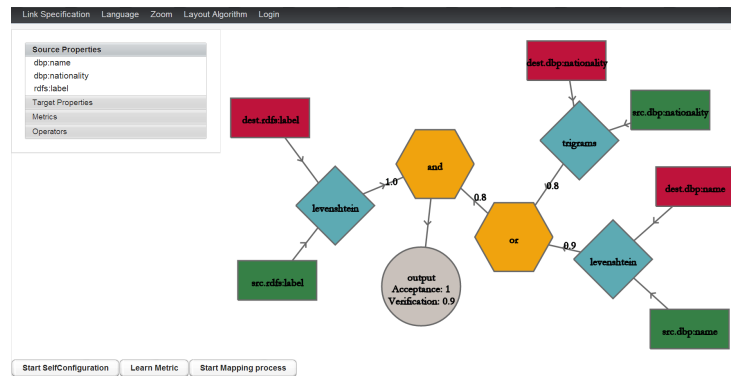


Figure 4: Screenshot of SAIM

³<http://saim.aksw.org>

3.4 Using Named Entity Recognition

The geographical information hidden in datatype properties is retrieved by using Named Entity Recognition. In the first version of GEOLIFT, we rely on the FOX framework. The FOX framework is a stateless and extensible framework that encompasses keyword extraction and named entity recognition. Its architecture consists of three layers as shown in Figure 5.

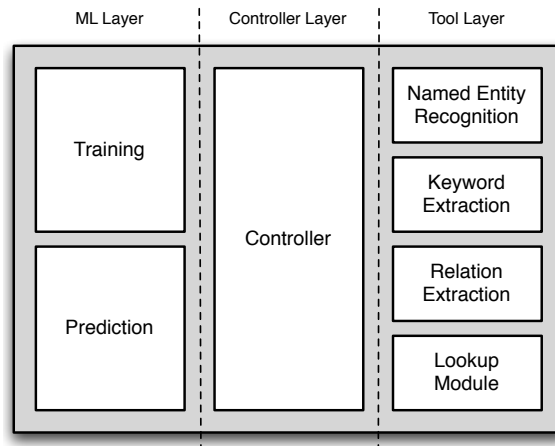


Figure 5: Architecture of the FOX framework.

FOX takes text or HTML as input. Here we use the objects of datatype properties, i.e., plain text. This data is sent to the *controller layer*, which implements the functionality necessary to clean the data, i.e., remove HTML and XML tags as well as further noise. Once the data has been cleaned, the controller layer begins with the orchestration of the tools in the *tool layer*. Each of the tools is assigned a thread from a thread pool, so as to maximize usage of multi-core CPUs. Every thread runs its tool and generates an event once it has completed its computation. In the event that a tool does not complete after a set time, the corresponding thread is terminated. So far, FOX integrates tools for KE, NER and RE. The KE is realized by tools such as KEA⁴ and the Yahoo Term Extraction service⁵. In addition, FOX integrates the Stanford Named Entity Recognizer⁶ [2], the Illinois Named Entity Tagger⁷ [10] and Alchemy⁸ for NER.

The results from the tool layer are forwarded to the *prediction module* of

⁴<http://www.nzdl.org/Kea/>

⁵<http://developer.yahoo.com/search/content/V1/termExtraction.html>

⁶<http://nlp.stanford.edu/software/CRF-NER.shtml>

⁷http://cogcomp.cs.illinois.edu/page/software_view/4

⁸<http://www.alchemyapi.com>

the *machine-learning layer*. The role of the prediction module is to generate FOX’s output based on the output the tools in FOX’s backend. For this purpose, it implements several ensemble learning techniques [1] with which it can combine the output of several tools. Currently, the prediction module carries out this combination by using a feed-forward neural network. The neural network inserted in FOX was trained by using 117 news articles. It reached 89.21% F-Score in an evaluation based on a ten-fold-cross-validation on NER, therewith outperforming even commercial systems such as Alchemy.

Once the neural network has combined the output of the tool and generated a better prediction of the named entities, the output of FOX is generated by using the vocabularies shown in Figure 6. These vocabularies extend the two broadly used vocabularies Annotea⁹ and Autotag¹⁰. In particular, we added the constructs explicated in the following:

- `scms:beginIndex` denotes the index in a literal value string at which a particular annotation or keyphrase begins;
- `scms:endIndex` stands for the index in a literal value string at which a particular annotation or keyphrase ends;
- `scms:means` marks the URI assigned to a named entity identified for an annotation;
- `scms:source` denotes the provenance of the annotation, i.e., the URI of the tool which computed the annotation or even the system ID of the person who curated or created the annotation and
- `scmsann` is the namespace for the annotation classes, i.e, location, person, organization and miscellaneous.

4 Developers’ Manual

4.1 GeoLift packages

GEOLIFT contains five basic *Java* packages:

1. `IO` package which deals with input/output operations using:
 - `Reader` class which handles all RDF reading process from file/URL/End-Point.

⁹<http://www.w3.org/2000/10/annotation-ns#>

¹⁰<http://commontag.org/ns#>

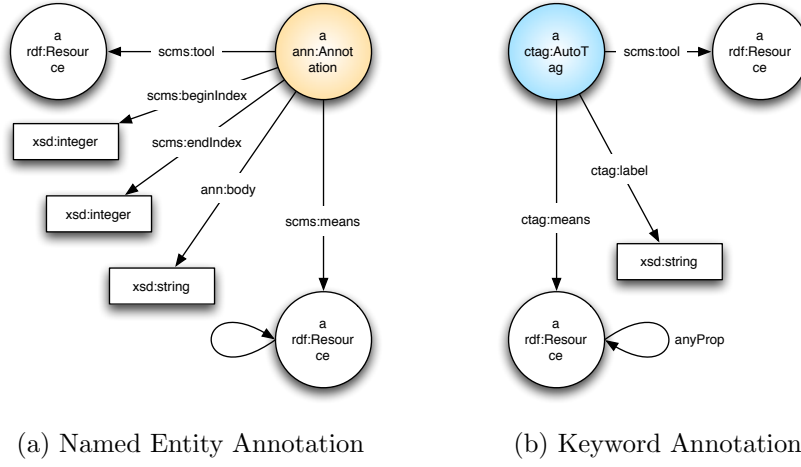


Figure 6: Vocabularies used by FOX for representing named entities (a) and keywords (b)

- **Writer** class which handles all the RDF writing process.
2. **Modules** package contains the **GeoLiftModule** interface which is implemented by all modules' classes. All modules can be created using the **ModuleFactory** class calling its **createModule()** method with the desired module's name as parameter. Also, the **getImplementations()** method returns all implemented modules. Currently, five modules are implemented:
 - **DereferencingModule** class which handles dereferencing geographical information extending process.
 - **LinkingModule** class which handles linking geographical information extending processes.
 - **NLPModule** class which handles named entity extraction process.
 - **ConformationModule** class which handles the conformation process.
 - **FilterModule** class which handles filter process.
 3. **Operators** package contains the **GeoLiftOperator** interface which is implemented by all operators' classes. All operators can be created using the **OperatorFactory** class calling its **createOperator()** method with the desired operator's name as parameter.
 - **SplitOperator** class to generate from 1 input dataset a set of $n \geq 2$ clone output datasets.

- **MergeOperator** class to generate one merged output dataset out of $n \geq 2$ input datasets.
- 4. **Helper package** containing helper classes used by GEOLIFT such as vocabularies classes.
- 5. **Workflow package** containing classes responsible for GEOLIFT overall execution process.

4.2 GeoLift Modules

By *Modules* we mean these artifacts in charge of generating geographical data based on RDF data. The input for such a module is an RDF dataset. The output is also an RDF dataset enriched with geographical information. Formally, a module can thus be regarded as a function $\mu : \mathcal{R} \rightarrow \mathcal{R}$, where \mathcal{R} is the set of all RDF datasets. Currently, GEOLIFT implements five modules: dereferencing, linking, NLP, conformation and filter.

All modules implement the **GeoLiftModule** interface methods

- **Model process()** method takes as input a *Jena* model and a Map of different parameters in form of ("parameterName", "parameterValue"), and as output all GEOLIFT module generates a Jena model also, this organization ease the usage of the modules in different workflows.
- **getParameters()**, returns list of all available module parameters.
- **getNecessaryParameters()**, returns list of all necessary module parameters that have no default values.
- **selfConfig(Model source, Model target)**, returns a map of parameters learned from the inputs (currently in progress).

Adding new modules to GEOLIFT is possible by implementing the **GeoLiftModule**.

4.2.1 Dereferencing Module

For datasets which contain similarity proprieties links (e.g. `owl:sameAs`), we deference all links from the dataset to other datasets by using a content negotiation on HTTP as shown in Figure 2. This returns a set of triples that needs to be filtered for relevant geographical information. Here, we use a predefined list of attributes that links to geographical information. Amongst others, we look for `geo:lat`, `geo:long`, `geo:lat_long`, `geo:line` and `geo:polygon`. The list of retrieved property values can be configured.

Table 1: Dereferencing module parameters

Parameter Key	Parameter value
<code>inputProperty<n></code>	List of interesting predicates to enrich the model, and their Objects' values. e.g. <code><"inputProperty1", "http://www.w3.org/2003/01/geo/wgs84_pos#lat"></code> . Mandatory parameter
<code>outputProperty<n></code>	The enriched output property. By default this parameter is set to <code>http://geoknow.org/ontology/relatedTo</code>
<code>useBlankNodes</code>	Use blank node in output dataset. By default, this parameter is set to false .

In this module, a *Java Jena* model and a list of interested predicates are given as inputs. This is done by iterating over the model's resources (dubbed as original resources) and for each of the original resource an extraction of the predicates' values (objects) that are in the form of URI is performed. These URIs (dubbed as dereferenced resources) are more filtered to be the resources used in *DBpedia*. The dereferenced resources are handled by a dereference operation in order to find the interested predicates list for them. Such predicates and their objects' values are fetched and added to the original resource to extend its information. Listing 1 provides a sample code showing how to use the `DereferencingModule` module:

4.2.2 Linking Module

Links to geographical resources do not occur in several knowledge bases. Here, we rely on the metrics implemented in the LIMES framework¹¹ [5, 4, 6] to link the resources in the input dataset with geographical datasets.

`LinkingModule` have two inputs: an input model and list of parameters (see table 2 for parameters details). The linking process starts by generating links between the input dataset model and another dataset as second partner. As said before, the linking process is done using LIMES. LIMES linking specification file is given as parameter. The output model of the `LinkingModule` is generated as the input model plus the generated links added with their original resources in the input model using some linking property such as `owl:samAs`. Listing 2 provides a sample code showing how to use the Linking module.

¹¹<http://limes.sf.net>

Listing 1: Code fragment to call the DereferencingModule class.

```

1  // Define DereferencingModule object
2  DereferencingModule u =
    ModuleFactory.createModule("dereferencing");
3  // Define parameters Map
4  Map<String, String> parameters = new HashMap<String, String>();
5  // Set parameters
6  parameters.put("predicate1", predicate1Value);
7  parameters.put("predicate2", predicate2Value);
8  // read input Model
9  Model model = Reader.readModel(datasetSource);
10 // Enrich the Model
11 Model resultedModel = u.process(model, parameters);
12 // Use the enriched model
13 Writer.writeModel(enrichedModel, "TTL", System.out);

```

Table 2: Linking module parameters

Parameter key	Parameter value
specFile	The path to specification file used for linking process, the original dataset to be enriched must be on the source dataset , e.g. linkingModuleData/linking/spec.xml.
linksFile	The path to links file resulted from the linking process. This file's path is the same as the one specified in LIME's specifications file as output file, e.g. linkingModuleData/linking/links.nt.
linksPart	Represents the original model's URI position as source/left or target/right in the linking specifications. Its value is either 'source' or 'target'.

Listing 2: Code fragment to call the `Linking` class.

```
1 // Define Linking object
2 Linking l = ModuleFactory.createModule("linking");
3 // Define parameters Map
4 Map<String, String> parameters = new HashMap<String, String>();
5 // Set parameters
6 parameters.put("datasetSource",datasetSourceValue);
7 parameters.put("specFilePath",specFilePathValue);
8 parameters.put("linksFilePath",linksPathValue);
9 parameters.put("linksPart",linksPartValue);
10 // read input Model
11 Model model = Reader.readModel(parameters.get("datasetSource"));
12 // Enrich the Model
13 model = l.process(model, parameters);
14 // Use the enriched model
15 Writer.writeModel(enrichedModel, "TTL", System.out);
```

4.2.3 NLP Module

The geographical information hidden in datatype properties is retrieved by using Named Entity Recognition (NER). GEOLIFT- as an expandable modeller framework - can inject any NER framework to implement the *NLP module*. In the current version of GEOLIFT, we rely on the FOX framework.

Table 3 provides details about the `NLPModule`'s parameters. A sample code demonstrating the `NLPModule` class is provided introduced in Listing 3.

4.2.4 Conformation Module

Change a specified source URI authority to a specified target URI authority, for example using source URI authority of `http://dbpedia.org` and target URI authority of `http://geolift.org` changes a resource like `http://dbpedia.org/Berlin` to `http://geolift.org/Berlin`. Moreover, the `ConformationModule` can be used to change a specific source properties to specific target properties. As an example, listing 4 introduce a code segment to manipulate the `ConformationModule` to change the source authority from `http://dbpedia.org` to the target property `http://geolift.org/Berlin`, and also change all occurrences of `RDF:label` to `SKOS:prefLabel`. Table 4 provides a full list of conformation module parameters.

Table 3: NLP module parameters

Parameter key	Parameter value
<code>literalProperty</code>	Literal property used by FOX for NER. If not set, the top ranked literal property will be pecked automatically by <code>LiteralPropertyRanker</code> sub-module, which ranks the lateral properties of a model according to the average size of each literal property divided by the number of instances of such property.
<code>addedGeoProperty</code>	property added to the input model with additional Geospatial knowledge through NLP. By default, this parameter is set to <i>gn:relatedTo</i> ¹¹
<code>useFoxLight</code>	<p>An implemented NER class name, either:</p> <ul style="list-style-type: none"> • <code>org.aksw.fox.nertools.NEROpenNLP</code> • <code>org.aksw.fox.nertools.NERIllinoisExtended</code> • <code>org.aksw.fox.nertools.NERIllinoisExtended</code> • <code>org.aksw.fox.nertools.NERBalie</code> • <code>org.aksw.fox.nertools.NERStanford</code> <p>By default this parameter is set to <code>OFF</code> in which all NER classes runs in parallel and a combined result will be returned, if this parameter is given with a wrong value, <code>org.aksw.fox.nertools.NERStanford</code> will be used.</p>
<code>NERType</code>	Force FOX to look for a specific NE’s types only. Available types are: <code>location</code> (default value), <code>person</code> , <code>organization</code> , and <code>all</code> to retrieve the all the previous three types.
<code>askEndPoint</code>	Ask the <i>DBpedia</i> endpoint for each location returned by FOX (setting it generates slower execution time but more accurate results). By default this parameter is set to <i>false</i>

Table 4: Conformation module parameters

Parameter key	Parameter value
<code>sourceSubjectAuthority</code>	Source URI to be replaced.
<code>targetSubjectAuthority</code>	Target URI to replace the sourceURI.
<code>sourceProperty<i></code>	i^{th} source property to be replaced by the i^{th} target property.
<code>targetProperty<i></code>	i^{th} target property to replace the i^{th} source property.

Listing 3: Code fragment to call the NLPModule class.

```
1 // Define NLPModule object
2 NLPModule geoEnricher= ModuleFactory.createModule("nlp");
3 // Define parameters Map
4 Map<String, String> parameters = new HashMap<String, String>();
5 // Set parameters
6 parameters.put("useFoxLight", "OFF");
7 parameters.put("askEndPoint", "false");
8 // read input Model
9 Model inputModel = Reader.readModel(inputFileValue);
10 // Enrich the Model
11 Model enrichedModel = geoEnricher.process(inputModel,
12     parameters);
13 // Use the enriched model
14 Writer.writeModel(enrichedModel, "TTL", System.out);
```

Listing 4: Code fragment to call the ConformationModule class.

```
1 // Define NLPModule object
2 ConformationModule geoEnricher=
3     ModuleFactory.createModule("conformation");
4 // Define parameters Map
5 Map<String, String> parameters = new HashMap<String, String>();
6 // Set parameters
7 parameters.put("sourceSubjectAuthority", "http://dbpedia.org");
8 parameters.put("targetSubjectAuthority", "http://geolift.org");
9 parameters.put("sourceProperty1",
10     "http://www.w3.org/1999/02/22-rdf-syntax-ns#label");
11 parameters.put("targetProperty1",
12     "http://www.w3.org/2004/02/skos/core#prefLabel");
13 // read input Model
14 Model inputModel = Reader.readModel(inputFileValue);
15 // Enrich the Model
16 Model enrichedModel = geoEnricher.process(inputModel,
17     parameters);
18 // Use the enriched model
19 Writer.writeModel(enrichedModel, "TTL", System.out);
```

Table 5: Filter module parameters

Parameter key	Parameter value
<code>triplesPattern</code>	Set of triple pattern to run against the input model of the filter module. By default, this parameter is set to <code>?s ?p ?o</code> . which generate the whole input model as output, changing the values of <code>?s</code> , <code>?p</code> and/or <code>?o</code> will restrict the output model.

Listing 5: Code fragment to call the `FilterModule` class.

```

1  // Define NLPModule object
2  FilterModule geoEnricher= ModuleFactory.createModule("filter");
3  // Define parameters Map
4  Map<String, String> parameters = new HashMap<String, String>();
5  // Set parameters
6  parameters.put("triplesPattern ", "?s
    <http://dbpedia.org/ontology/abstract> ?o");
7  parameters.put("askEndPoint", "false");
8  // read input Model
9  Model inputModel = Reader.readModel(inputFileValue);
10 // Enrich the Model
11 Model enrichedModel = geoEnricher.process(inputModel,
    parameters);
12 // Use the enriched model
13 Writer.writeModel(enrichedModel, "TTL", System.out);

```

4.2.5 Filter Module

Runs a set of triples patterns' against an input model to filter some triples out of it and export them to an output model. For example running triple pattern `"?s <http://dbpedia.org/ontology/abstract> ?o"` against an input model containing `"http://dbpedia.org/resource/Berlin"` will generate output model containing only Berlin's abstracts of *DBpedia*. Listing 5 demonstrate the filter module usage. table 5 provides a full list of the filter module parameters.

4.3 GeoLift Operators

The idea behind operators is to enable users to define a workflow for processing their input dataset. GEOLIFT operators combine the enrichment modules and allow defining a workflow for processing information. Note that

the format of the input and output of operator is identical. Thus, the user is empowered to create workflows of arbitrary complexity by simply connecting modules and/or operators. Formally, an operator can be regarded as a function $\varphi : \mathcal{R}^* \rightarrow \mathcal{R}^*$.

All operators classes implement the `GeoLiftOperator` interface's two methods `getParameters()` and `process()`, where the `getParameters()` method returns a list of its module input parameters, and the `process()` method takes as input a list of *Jena* models and a *Map* of different parameters in form of ("parameterName", "parameterValue"). Each of the GEOLIFT operators outputs a list of *Jena* models. This organization ease the usage of the operators in different workflows. Adding new operators to GEOLIFT is possible by implementing the `GeoLiftOperator`.

4.3.1 Split Operator

The idea behind the split operator is to enable parallel execution of different modules in the same dataset. The split operator takes one dataset as input and produces $n \geq 2$ output datasets, which are all identical to the input dataset. Each of the output datasets of the split operator has its own workflow (as to be input to any other module or operator). Thus, GEOLIFT is able to execute all workflows of output datasets in parallel.

4.3.2 Merge Operator

The idea behind the merge operator is to enable combining datasets. The merge operator takes a set of $n \geq 2$ input datasets and merges them into one output dataset containing all the input datasets' triples. As in case of split operator, the merged output dataset has its own workflow (as to be input to any other module or operator).

5 GeoLift RDF Specification Paradigm

In the current version of GEOLIFT we introduce our new RDF based specification paradigm. The main idea behind this new paradigm is to enable the processing execution of specifications in an efficient way. To this end, we first decided to use RDF as language for the specification. This has the main advantage of allowing for creating specification repositories which can be queried easily with the aim of retrieving accurate specifications for the use cases at hand. Moreover, extensions of the specification language do not require a change of the specification language due to the intrinsic extensibility

of ontologies. The third reason for choosing RDF as language for specifications is that we can easily check the specification for correctness by using a reasoner, as the specification ontology allows for specifying the restrictions that specifications must abide by.

introduces an example of RDF specification file presented in turtle serialization, which will be used as a guiding example through the next subsections. For the sake of clarity, we represent the configuration file in form of graph in figure 7.

5.1 GeoLift Execution Workflow

First of all, GEOLIFT determines the set of output datasets $D = \{d_1, \dots, d_n\}$, i.e. datasets which are included as output of some modules/operators but not as input to any other modules/operators. In our example it is only d_8 . Then, for each dataset d_i in D , GEOLIFT tries to find d_i either trivially from file/URI/endpoint or recursively by solving for modules/operators that generate d_i as output.

Going back to our example, GEOLIFT first tries to read d_8 trivially but it fails as there is no direct way to read it. Then, GEOLIFT recursively goes for solving the **Conformation** module as it generates d_8 as output. This recursive procedure continues until GEOLIFT reads the dataset d_1 from the endpoint. Afterwards, going back in the recursion stack GEOLIFT produces the triples of d_8 .

5.1.1 RDF Specification Resources

GEOLIFT RDF specification file may contain four main resource types (dataset, module, operator and parameter). In the following we will describe each of them along with our example shown in listing 6.

- **Dataset Resource** As its name implies, a Dataset resource represents a dataset. For an example see listing 6 (lines 4:6), in which the RDF specification represents a dataset with URI `http://dbpedia.org/resource/Berlin` and endpoint `http://dbpedia.org/sparql`. GEOLIFT extracts the CBD of the *Berlin* resource from *DBpedia* endpoint.

Beside defining a resource as a dataset (e.g. listing listing 6 (lines 4)), a dataset resource may also contain the following predicates:

- `rdfs:label` setting label for the dataset.
- `:hasUri` setting URI of the dataset resource, to read resource through content negotiation (e.g. listing listing 6 (line 5)).

Listing 6: Example of RDF configuration file.

```

1 @prefix : <http://geoknow.org/specsontology/> .
2 @prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#> .
3 @prefix geo: <http://www.w3.org/2003/01/geo/wgs84_pos#> .
4 :d1      a          :Dataset ;
5          :hasUri      <http://dbpedia.org/resource/Berlin> ;
6          :fromEndPoint <http://dbpedia.org/sparql> .
7 :d2      a          :Dataset .
8 :d3      a          :Dataset .
9 :d4      a          :Dataset .
10 :d5     a          :Dataset .
11 :d6     a          :Dataset .
12 :d7     a          :Dataset .
13 :d8     a          :Dataset ;
14         :outputFile  "GeoLiftBerlin.ttl" ;
15         :outputFormat "Turtle" .
16 :deref   a          :Module, :DereferencingModule ;
17         rdfs:label    "Dereferencing module" ;
18         :hasInput     :d1 ;
19         :hasOutput     :d2 ;
20         :hasParameter :derefParam1 .
21 :derefParam1 a :ModuleParameter, :DereferencingModuleParameter ;
22             :hasKey      "inputProperty1" ;
23             :hasValue     geo:lat .
24 :split    a          :Operator, :SplitOperator ;
25         rdfs:label    "Split operator" ;
26         :hasInput     :d2 ;
27         :hasOutput     :d3, :d4 .
28 :nlp      a          :Module, :NLPModule ;
29         rdfs:label    "NLP module" ;
30         :hasInput     :d3 ;
31         :hasOutput     :d5 ;
32         :hasParameter :nlpPram1, :nlpPram2 .
33 :nlpPram1 a          :ModuleParameter, :NLPModuleParameter ;
34         :hasKey      "useFoxLight" ;
35         :hasValue     "OFF" .
36 :nlpPram2 a          :ModuleParameter, :NLPModuleParameter ;
37         :hasKey      "askEndPoint" ;
38         :hasValue     false .
39 :filter   a          :Module, :FilterModule ;
40         rdfs:label    "Filter module" ;
41         :hasInput     :d4 ;
42         :hasOutput     :d6 ;
43         :hasParameter :filterPram1 .
44 :filterPram1 a :ModuleParameter, :NLPModuleParameter ;
45             :hasKey      "triplesPattern" ;
46             :hasValue     "?s <http://dbpedia.org/ontology/abstract> ?o".
47 :merge    a          :Operator, :MergeOperator ;
48         rdfs:label    "Merge operator" ;
49         :hasInput     :d6, :d5 ;
50         :hasOutput     :d7 .
51 :conform  a          :Module, :ConformationModule ;
52         rdfs:label    "Conformation module" ;
53         :hasInput     :d7 ;
54         :hasOutput     :d8 ;
55         :hasParameter :conformPram1, :conformPram2 .
56 :conformPram1 a :ModuleParameter, :NLPModuleParameter ;
57             :hasKey      "sourceURI" ;
58             :hasValue     "http://dbpedia.org" .
59 :conformPram2 a :ModuleParameter, :NLPModuleParameter ;
60             :hasKey      "targetURI" ;
61             :hasValue     "http://geolift.org" .

```

- `:fromEndPoint` end point to read the CBD from. *Note: must be used in conjunction with `:hasUri` predicate, otherwise an error will be generated by GEOLIFT. (e.g. listing 6 (line 6)).*
 - `:inputFile` input file to load the dataset from.
 - `:outputFile` output file to save the dataset to (e.g. listing 6 (lines 14)).
 - `:outputFormat` output format to save the dataset in it (e.g. listing 6 (lines 14)), Turtle format will be used as default format if this property is not set.
- **Module Resource** Beside defining a resource as a module (e.g. listing 6 (line 16)), a Module resource may also contain the following predicates:
 - `rdfs:label` setting label for the module (e.g. listing 6 (line 17)).
 - `:hasInput` setting input datasets for the module (e.g. listing 6 (line 18)).
 - `:hasOutput` setting output datasets for the module (e.g. listing 6 (line 19)).
 - `:hasParameter` setting parameters for the module (e.g. listing 6 (line 20)).
 - **Operator Resource** Beside defining a resource as an operator (e.g. listing 6 (line 24)), a Module resource may also contain the following predicates:
 - `rdfs:label` setting a label for the operator (e.g. listing 6 (line 25)).
 - `:hasInput` setting input datasets for the operator (e.g. listing 6 (line 26)).
 - `:hasOutput` setting output datasets for the operator (e.g. listing 6 (line 27)).
 - `:hasParameter` setting parameters for the operator.
 - **Parameter Resource** Beside defining a resource as a parameter (e.g. listing 6 (line 21)), a Module resource may also contain the following predicates:
 - `rdfs:label` setting label for the parameter.

- `:hasKey` setting the parameter key (*e.g. listing 6 (line 22)*).
- `:HasValue` setting the parameter value (*e.g. listing 6 (lines 23)*).

The values of (key, values) pairs are modules/operators dependent ¹².

6 Running GeoLift From Command-Line

The GEOLIFT can be directly executed from command-line using the provided `geolift.jar`¹³ file. Simply, provide the RDf configuration file as the only one parameter for the GEOLIFT jar file.

Example : `geolift.jar src/main/resources/workflow/config.ttl`

7 Conclusions

In this manual, we presented the GEOLIFT component for enriching RDF datasets with geo-spatial data. In future work, we aim to implement a graphical user interface on top of GEOLIFT to enable users to specify their workflows graphically. Moreover, we aim to implement workflow checking functionality.

References

- [1] Thomas G. Dietterich. Ensemble methods in machine learning. In *MCS*, pages 1–15, London, UK, 2000. Springer-Verlag.
- [2] J. Finkel, T. Grenager, and C. Manning. Incorporating non-local information into information extraction systems by gibbs sampling. In *ACL*, pages 363–370, 2005.
- [3] Axel-Cyrille Ngonga Ngomo. Link discovery with guaranteed reduction ratio in affine spaces with minkowski measures. In *Proceedings of ISWC*, 2012.
- [4] Axel-Cyrille Ngonga Ngomo. On link discovery using a hybrid approach. *Journal on Data Semantics*, 1:203 – 217, December 2012.

¹²For more details about (key, values) pairs of each module/operator, see GEOLIFT user manual at <https://github.com/GeoKnow/GeoLift/blob/master/GeoLiftManual/GeoLiftManual.pdf>

¹³the `geolift.jar` file is in the `jars` folder of the project

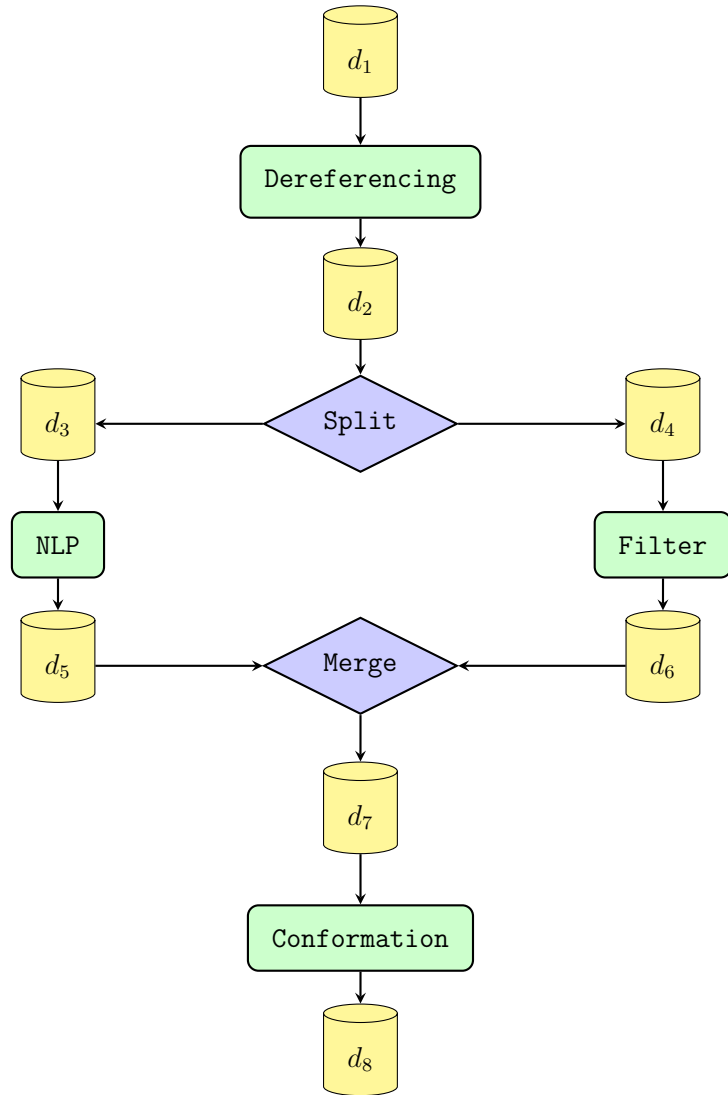


Figure 7: Graph representation of configuration file presented by listing 6

- [5] Axel-Cyrille Ngonga Ngomo and Sren Auer. A time-efficient approach for large-scale link discovery on the web of data. In *IJCAI*, page 2011, 2011.
- [6] Axel-Cyrille Ngonga Ngomo, Lars Kolb, Norman Heino, Michael Hartung, Sören Auer, and Erhard Rahm. When to reach for the cloud: Using parallel hardware for link discovery. In *Proceedings of ESCW*, 2013.
- [7] Axel-Cyrille Ngonga Ngomo, Jens Lehmann, Sören Auer, and Konrad Höffner. Raven – active learning of link specifications. In *Proceedings of OM@ISWC*, 2011.
- [8] Axel-Cyrille Ngonga Ngomo and Klaus Lyko. Eagle: Efficient active learning of link specifications using genetic programming. In *Proceedings of ESWC*, 2012.
- [9] Axel-Cyrille Ngonga Ngomo, Klaus Lyko, and Victor Christen. Coala – correlation-aware active learning of link specifications. In *Proceedings of ESWC*, 2013.
- [10] Lev Ratinov and Dan Roth. Design challenges and misconceptions in named entity recognition. In *CONLL*, pages 147–155, 2009.