ICT Seventh Framework Programme (ICT FP7)

Grant Agreement No: 318497

Data Intensive Techniques to Boost the Real – Time Performance of Global Agricultural Data Infrastructures

Source Metadata Editing: Guidelines and Tools

This document provides guidelines and best practices when authoring the data source descriptions needed to add a data source to a SemaGrow deployment. It also presents alternatives for carrying out this authoring task, providing guidance about when to prefer each alternative.

One of these alternatives is using the ELEON Annotation Environment, an RDF metadata editor customized to the purpose of editing data source descriptions. The second part of this document is a user guide for ELEON.

TABLE OF CONTENTS

1. INTRODUCTION

1.1 The SemaGrow Stack

Every deployment of the *SemaGrow Stack* presents a single SPARQL endpoint that federates a number of SPARQL endpoints. These federated SPARQL endpoints, the *data sources* of the deployment, are publicly available and independently maintained and do not need to by modified in any way in order to participate in the federation; in fact, they do not even need to be aware of the fact that they have been included in a SemaGrow federation.

SemaGrow aims to offer the most efficient distributed querying solution that can be achieved without controlling the way data is distributed between sources and, in general, without having the responsibility to centrally manage the data sources of the federation. In other words, the *SemaGrow Stack* exposes a SPARQL 1.1 endpoint[1] that:

- Supports the SPARQL recommendation's federated querying capabilities. Pushing beyond the recommendation and the state of the art, the *SemaGrow Stack* is able to automatically break up queries into *query fragments* and to distribute these query fragments among the federated endpoints without requiring that the user tells the system where to dispatch each fragment using the SERVICE keyword.
- Does *not* support INSERT, DELETE and, in general, does not provide any means of updating the data held in the federated sources.

In order to be able to intelligently decide about how to distribute queries, *SemaGrow* depends on some information of the data exposed by each member of its federation. Although the SemaGrow Stack is able to automatically make some educated guesses about the contents of an endpoint by analysing its response to user queries, some information should be provided manually: at a minimum the URL of the endpoint that is to be federated should be provided by the administrator of the SemaGrow deployment.

This document explains the kinds of information that constitutes useful data source descriptions and provides guidelines and best practices when authoring data source descriptions. It also presents alternatives for carrying out this authoring task, providing guidance about when to prefer each alternative. One of these alternatives is using the *ELEON Annotation Environment*, an RDF metadata editor customized to the purpose of editing data source descriptions. The final chapter of this document is a user guide for ELEON.

1.2 Core Concepts

There are four levels at which data sources can be described:

- A title by which to refer to the endpoint and its URL. This is absolute minimum that must be provided by the human operator.
- *Schema-level annotations,* such as the properties and classes instantiated within the dataset. Although this information can be automatically discovered while using the endpoint, it can also be easily provided by a human operator. When using the ELEON environment, this is as easy as providing the schema specification in RDF, which ELEON analyses to extract schema-level annotations.
- *Schema-level cardinalities*, such the number of triples in the overall dataset and how it breaks down into the number of triples for each predicate in the dataset schema, and the number of triples having a subject (or object) within each class in the dataset schema. A data provider can easily direct measure such statistics, although this might be harder to maintain for evolving datasets. They can also be automatically estimated by the SemaGrow Stack while using the endpoint, although accuracy of the estimation depends heavily on the query workload.

[1] *SPARQL 1.1 Federated Query*, W3C Recommendation, 21 March 2013. http://www.w3.org/TR/sparql11-federated-query

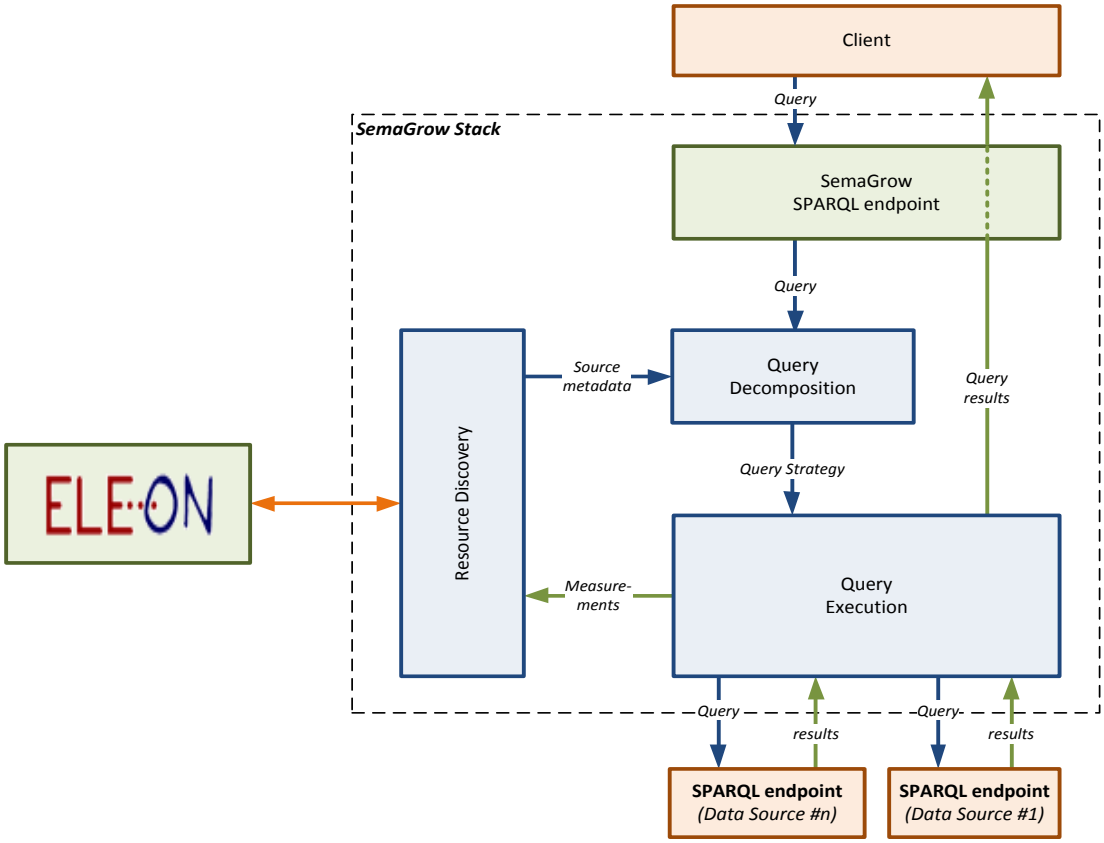

Figure 1: Semagrow Stack architecture, showing interactions involving data source annotations.

- *Instance-level annotations*, akin to database *histograms*, detailed statistics about not just predicate and class cardinalities, but also value ranges and their cardinalities. Such detailed statistics also include the *selectivity* of joining different triple patterns, including the selectivity of joins across different endpoints. Such detailed statistics cannot be reasonably provided by human operators, and are automatically estimated.

Based on this metadata, the *Query Decomposition* engine of the *SemaGrow Stack* can make intelligent decisions about how to distribute a query among the members of the federation and how to plan query execution. Based on measurements received during query execution, the system can estimate this statistics. Finally, annotations can be serialized into (and deserialized from) RDF for the purpose of human operator inspection and editing. Figure 1 provides an overview of these interactions.

Regardless of how such annotations are internally represented in the SemaGrow Stack, SemaGrow has defined *Sevod*, an RDF schema for the serialization of data source annotations. Sevod will be presented in more detail below (Chapter 2). The distinction should be noted between:

- The *data schema or schemas* used in the datasets themselves; and
- The *annotation schema* used in order to describe datasets (including to specify what data schemas are used in them). SemaGrow uses the Sevod as annotation schema.

In Sevod data repositories (and collections comprising multiple data repositories) are organized as a hierarchy of *datasets* and their subsets: a dataset contains all of the RDF triples contained in all of its subsets; these subsets are a dataset in their own right, and can be broken down into smaller subsets. Each dataset is annotated with properties pertaining to two aspects:

- What's in it, in terms of predicates used, the class of its subjects or objects, the range of the values (objects) for a particular predicate, the regular expression for the URIs of its subjects, or similar annotations that specify which triples can be found in a dataset. In this document, and following standard parlance in database histograms, we will call these the *bucket* that contains the dataset.

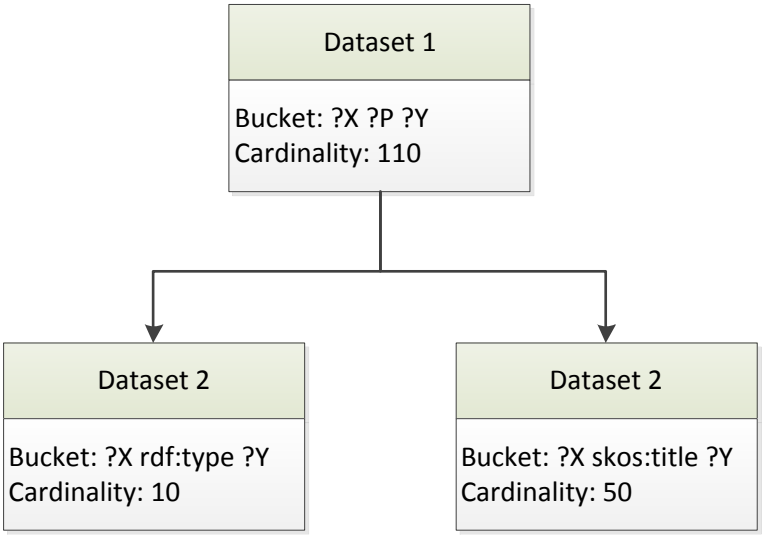

Figure 2: Dataset inclusion example

- How many of those there are, that is, the number of triples that match the criteria above and are, thus, within this dataset. We will call these the *statistics* of the dataset. We might sometimes refer to the statistics of a bucket, to mean the statistics of the dataset contained in a bucket.

Each dataset is a node in a tree-like inclusion hierarchy,[2] where parents contain at least all of the triples contained in their children. The statistics reported for each dataset are inclusive of those of its children and *not* the residue. So, for example, the parent *Dataset 1* in Figure 2 contains 110 triples and *not* 170 triples.

Sevod is described in more detail in Chapter 2 of this document, including best practices regarding dataset annotation. Chapter 3 presents simple configuration files and the ELEON Authoring Environment, both aiming to provide an easier alternative for producing *Sevod* descriptions than directly authoring RDF code.

[2] Multiple inheritance is supported, but it is very rarely useful when annotating datasets.

2. DATA SOURCE ANNOTATION SCHEMA

2.1 Overview

The data model of the SemaGrow data source annotations has been formalized as the *Sevod* RDFS vocabulary. Sevod extends the *Vocabulary of Interlinked Datasets (VoID).*[3] VoID properties provide information about size and semantics, such as the number of triples in a dataset, the number of distinct entities mentioned in these triples, the classes that characterize these entities, etc. Sevod extends VoID with more detailed information mostly addressing how different datasets can be joined in responses to queries that combine information from multiple data sources. The top Sevod classes and their relationship to their VoID super-classes is shown in Figure 3.

In VoID and Sevod, as well as in their visual presentation in ELEON (cf. Chapter 3), there is no class distinction between the different levels of datasets in the hierarchy: a dataset can be anything from a huge collection to a handful of triples that is a tiny part of the data served by a data source. Conceptually, however, we will make the following distinctions:

- We will call *data source* a dataset that is served by a SPARQL endpoint. All data in subsets of a dataset are assumed to be accessible via the data source's endpoint. The importance of data sources is that they are the entities that can be accessed by the query engine; all information about their subsets is only relevant for deciding about an optimal query execution plan, but such a plan can only decide about how to distribute queries among the endpoints of the federation and cannot specify the query execution plan decided internally by each endpoint's query execution engine.
- We will call *collection* a dataset that comprises several data sources that are known by collective name and that are provided by a single creator entity, follow the same data schemas, or have some other unifying attributes. The importance of collections is that they offer a human-readable title and other provenance information that can be used to prefer or exclude collections from the data considered for a given query.
- We will call *subset* or *dataset* a dataset that is part of a data source.

We will call a dataset *complete* if it either (a) has no subsets or (b) among its direct subsets, there is a set of subsets that form a partitioning of the dataset. The importance of *complete data sources* is that besides knowing what can be found there, we can also infer that some resources *cannot* be possibly found at a data source.

Although this three-level organization is not foreseen by either VoID or Sevod, and neither enforced by ELEON, it should be considered a *best practice recommendation* for maximizing the usefulness of the descriptions for the purpose of query engine optimization that:

- Collections should have a meaningful title. Data sources should have a meaningful title, if not part of a collection.
- Data sources should be complete datasets, even if their subsets are not.

We will now proceed to define in more detail the Sevod vocabulary.

2.2 Dataset hierarchy

Database histograms capture statistics on cardinality and selectivity for the purpose of query optimization. This statistics is organized as an inclusion hierarchy of *buckets*, where each bucket stores:

1. The name of an attribute and a range of values for this attribute. In multi-dimensional histograms, a bucket is characterized by multiple (attribute, value range) pairs.
2. The cardinality of this attribute range, that is, the number of tuples where the given attribute has a value within the given range.

Sevod uses the void:Dataset class and the void:subset property to represent the dataset of a bucket and the hierarchical organization between buckets.

[3] cf. K. Alexander, R. Cyganiak et al., *Describing Linked Datasets with the VoID Vocabulary*. W3C Interest Group Note, 3 March 2011. Available at http://www.w3.org/TR/void

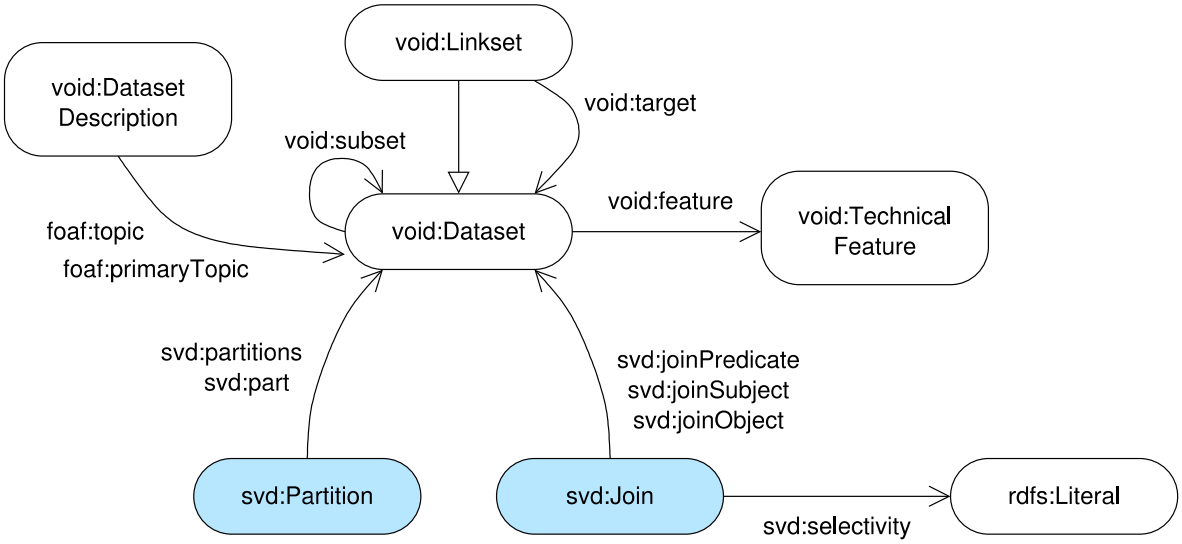

Figure 3: Sevod top classes and relationship to VoID

2.3 Partitioning

Histogram buckets are captured by void:Dataset instances, which can be hierarchically organized using the void:subset property.VoID, however, does not allow the distinction between dataset partitionings and overlapping datasets. To cover this requirement, Sevod introduces *partitions* as follows:

Definition: sevod:Partition denotes that a set of void:Dataset instances are a *partition* of another void:Dataset instance. This is done by using the property sevod:part to link the sevod:Partition instance with the instances that make up the partition and the property sevod:partitions to link it with the instance that is partitioned by them.

Definition: svd:partitions is the *functional* property that links a sevod:Partition instance with the void:Dataset for which it is a partition.

Definition: The sevod:part property links a sevod:Partition instance with each of the void:Dataset instances that make up the partition. All fillers of this propoerty must also be fillers of the void:subset property of the void:Dataset instance that fills the sevod:Partition instance's sevod:partitions property.

Figure 4 gives a characteristic example from the Trees4Future metadata.

2.4 Multi-Dimensional Buckets

A further characteristic of database histograms that VoID cannot capture is *multi-dimensional buckets*, corresponding in RDF to buckets that describe *joins* of triple patterns:

Definition: A sevod:Join instance connects two void:Dataset instances with an integer value that is (or estimates or approximates) the *selectivity* of the join of these datasets. The triple element that is joined is denoted by the specific subpropery of the sevod:joins property used to link the sevod:Join instance with the void:Dataset instances.

Definition: The sevod:joins property links a sevod:Join instance with the void:Dataset instances that are joined.

Definition: The sevod: joinSubject property links a sevod:Join instance j with a void:Dataset instance f iff

1. there is also a triple j svd:joins e, where e is a void:Dataset instance; and
2. j denotes the join of d on the subject of its triples with e on the element of its triples denoted by the specific subproperty of devod:joins used in the j svd:joins e triple above.

Definition: The sevod:joinPredicate property links a sevod:Join instance j with a void:Dataset instance d iff

1. there is also a triple j sevod:joins e, where e is a void:Dataset instance; and
2. j denotes the join of d on the predicate of its triples with e on the element of its triples denoted by the specific subproperty of svd:joins used in the j sevod:joins e triple above

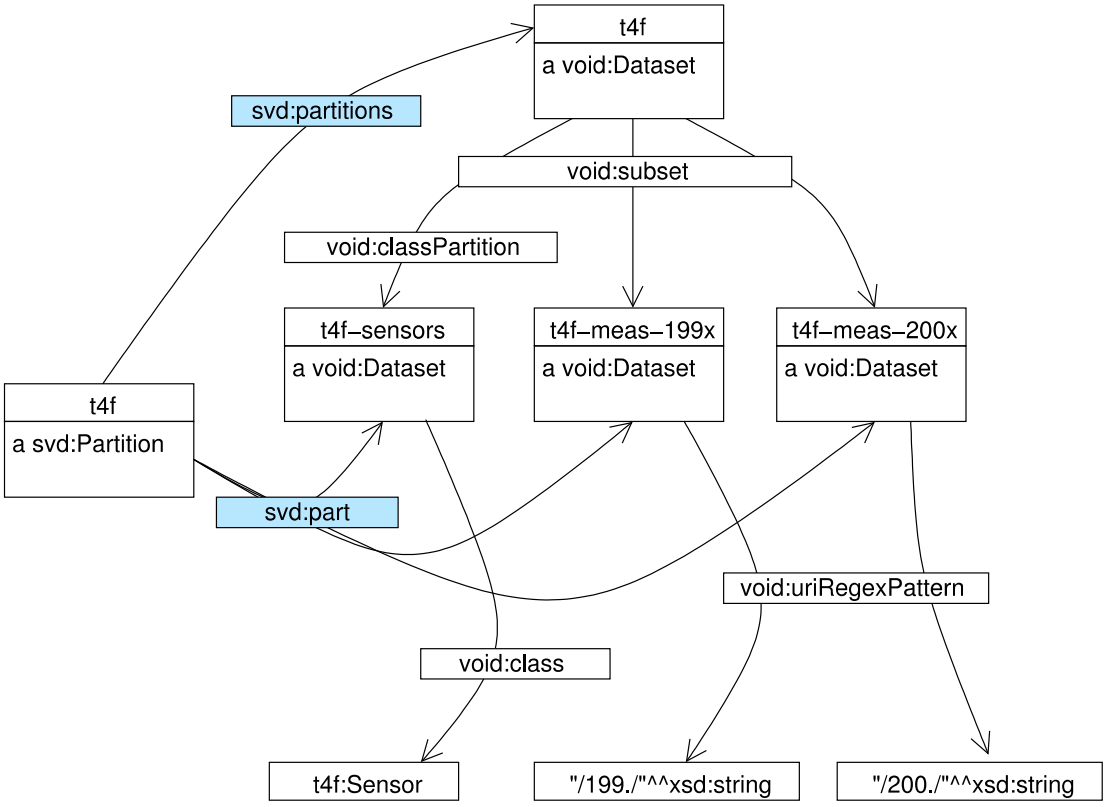

Figure 4: Example usage of sevod:Partition, taken from Trees4Future

Definition: The sevod:joinObject property links a sevod:Join instance j with a void:Dataset instance d iff

1. there is also a triple j sevod:joins e, where e is a void:Dataset instance; and
2. j denotes the join of d on the object of its triples with e on the element of its triples denoted by the specific subproperty of sevod:joins used in the j svd:joins e triple above
3. **Definition:** The sevod:selectivity property links a sevod:Join instance with the instance of sevod:SelectivityValue that is (or estimates or approximates) the selectivity of the join denoted by the sevod:Join instance.

Finally, bucket statistics do not need to be scalar values, but may also be more complex representations of constraints over or distributions of scalar values that are not precisely known. In order to cover this requirement, we define the range of the sevod:selectivity property to be a class instead of simple numerical fillers. In this manner, we encapsulate statistics under a class that can be extended to cover application-specific requirements. In order to ensure compatibility, we further require that instances of this sevod:SelectivityValue class must have an rdf:value property and that this property has as value an xsd:integer. Other, application specific, properties may be defined as needed for extensions of this class.

Definition: Instances of the sevod:SelectivityValue class denote an exact, estimated, or approximated measurement. sevod:SelectivityValue instances must have the rdf:value property with an filler that is an xsd:integer that is either the value itself (if the measurement is exact and certain) or a value that is appropriate to use by applications that do not take into account uncertainty or approximation parameters. Where appropriate, uncertainty or approximation parameters are given by application-specific properties.

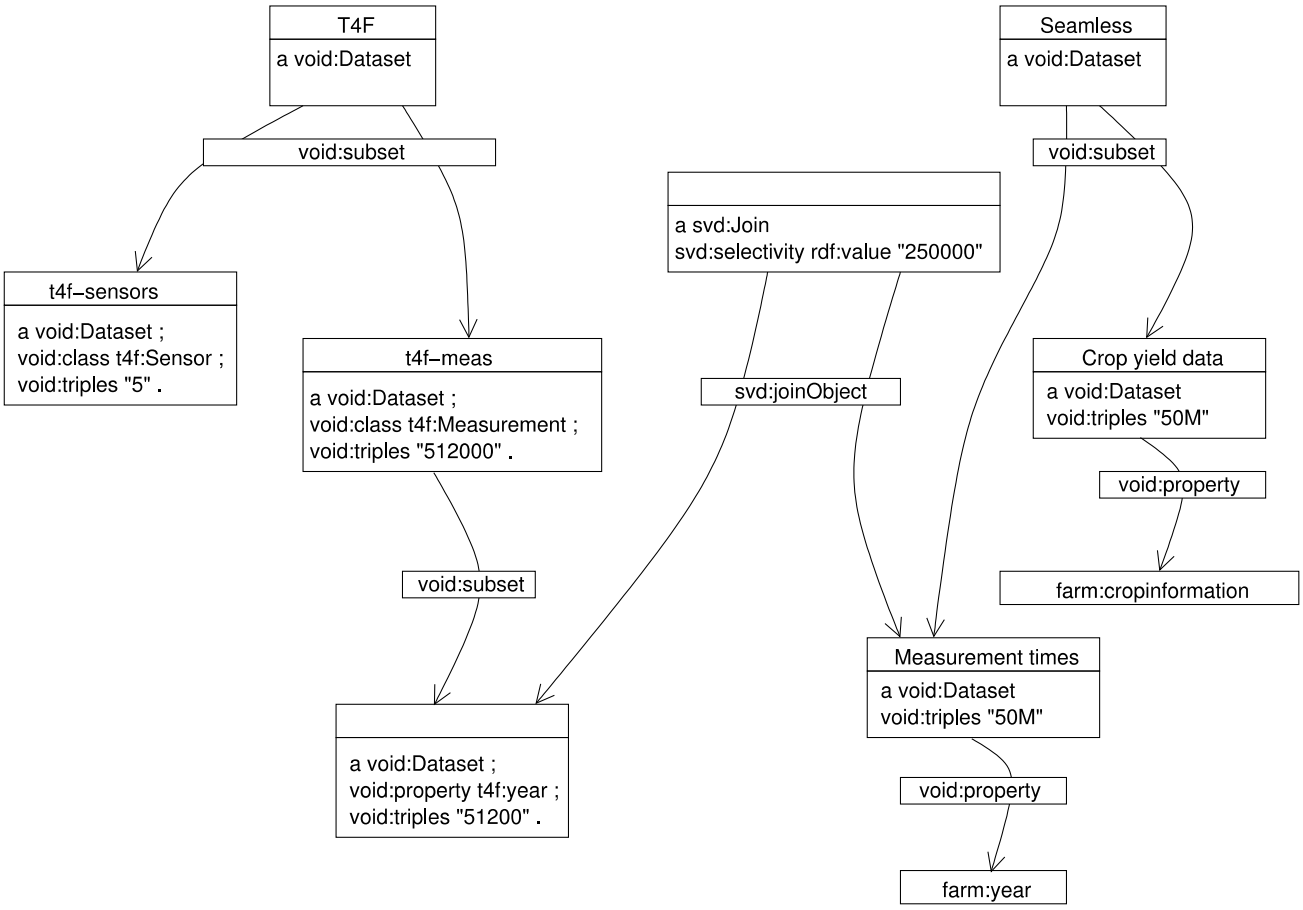

Figure 5: Example usage of sevod:Join

2.5 List of Sevod Vocabulary Items

The complete list of Sevod vocabulary items and their RDFS axiomatization is as follows:

```
@prefix svd:  <http://www.semagrow.eu/2014/sevod#>
@prefix void: <http://rdfs.org/ns/void#>
@prefix rdf:  <http://www.w3.org/1999/02/22-rdf-syntax-ns#>
@prefix rdfs: <http://www.w3.org/2000/01/rdf-schema#>

svd:subjectUriRegexPattern
      rdf:subPropertyOf void:uriRegexPattern .
svd:subjectClass
      rdf:subPropertyOf void:class .
svd:subjectVocabulary
      rdf:subPropertyOf void:vocabulary .
svd:objectRange rdf:type rdf:Property .
svd:objectUriRegexPattern
      rdf:subPropertyOf svd:objectRange ;
      rdf:subPropertyOf void:uriRegexPattern .
svd:objectClass
      rdf:subPropertyOf void:class .
```

```
svd:objectVocabulary
      rdf:subPropertyOf void:class .
svd:Interval rdf:type rdf:Class .
svd:from rdf:type rdf:Property ;
        rdf:domain svd:Interval .
svd:to rdf:type rdf:Property ;
        rdf:domain svd:Interval .
svd:IntegerInterval
      rdf:subClassOf svd:Interval .
svd:RealInterval
      rdf:subClassOf svd:Interval .
svd:DateInterval
      rdf:subClassOf svd:Interval .
svd:objectInterval
      rdf:subPropertyOf svd:objectRange ;
      rdf:range svd:Interval .
svd:intInterval
      rdf:subPropertyOf svd:objectInterval ;
      rdf:range svd:IntegerInterval .
svd:dateInterval
      rdf:subPropertyOf svd:objectInterval ;
      rdf:range svd:DateInterval .
svd:realInterval
      rdf:subPropertyOf svd:objectInterval ;
      rdf:range svd:RealInterval .
svd:objectPartition
      rdf:subPropertyOf void:subsetOf .
svd:UncertainValue rdf:type rdfs:Class .
svd:selectivity
      rdfs:range svd:UncertainValue .
svd:cardinality
      rdfs:range <http://www.w3.org/2001/XMLSchema#nonNegativeInteger> .
svd:Join rdf:type rdfs:Class .
svd:joinsDataset
      rdf:domain svd:Join ;
      rdf:range void:Dataset .
svd:joinsSubject
      rdf:subPropertyOf svd:joinsDataset .
svd:joinsPredicate
      rdf:subPropertyOf svd:joinsDataset .
svd:joinsObject
    rdf:subPropertyOf svd:joinsDataset .
```

3. Source Metadata Editing

3.1 Configuration Files

Simple data source annotations can be provided by directly editing the SemaGrow Stack configuration, which is the RDF/TTL file located at `/etc/default/Semagrow/metadata.ttl`

The only obligatory properties of a dataset are dct:title and void:sparqlEndpoint, so a very simple example could be:[4]

```
<http://example.org/id/datasource01>
	dct:title "EU Open Data Portal" ;
	dct:description "The EU Open Data Portal provides, via a metadata catalogue, a single point of access to data of the EU institutions, agencies and bodies for anyone to reuse." ;
	rdf:seeAlso <http://open-data.europa.eu> ;
	void:sparqlEndpoint <http://open-data.europa.eu/sparqlep> .
```

A slightly more complicated example can provide some top-level statistics as well as sub-datasets and their statistics.

```
<http://example.org/id/datasource02>
	dct:title "AGRIS" ;
	void:sparqlEndpoint <http://202.45.139.84:10035/catalogs/fao/repositories/agris> ;
	dct:description "AGRIS contains more than 7 million bibliographic references on agricultural research and technology & links to related data resources on the Web, like DBPedia, World Bank, Nature, FAO Fisheries and FAO Country profiles." ;
	rdf:seeAlso <http://agris.fao.org> ;
	void:sparqlEndpoint <http://open-data.europa.eu/sparqlep> ;
	void:triples "189924581"^^xsd:int ;
	void:distinctSubjects "30789981"^^xsd:int ;
	void:vocabulary <http://purl.org/dc/terms/> , <http://purl.org/ontology/bibo/> ,
		<http://xmlns.com/foaf/0.1/> ;
	void:subset [
		void:property <http://purl.org/dc/terms/issued> ;
		sevod:objectRegexPattern "199[0-9]"^^xsd:string ;
		void:distinctObjects "1633"^^xsd:int .
	] .
```

The sevod:objectRegexPattern property applies to the lexical value of literals of any type and to the string representation of the URI of resources. For numerical types only, one can also specify ranges using the svd:objectRange property filled with instances of sevod:Interval, as in this example:

```
<http://example.org/id/datasource02> void:subset [
	void:property <http://semagrow.eu/rdf/cleandates/yearIssued> ;
	sevod:objectRange [ sevod:from "1985"^^xsd:gYear; sevod:to "1994"^^xsd:gYear ] .
] .
```

[4] We use TTL notation and assume the following prefixes: rdf: (http://www.w3.org/1999/02/22-rdf-syntax-ns#), xsd: (http://www.w3.org/2001/XMLSchema#), dct: (http://purl.org/dc/terms/), void: (http://rdfs.org/ns/void#), sevod: (http://rdf.iit.demokritos.gr/2013/)

3.2 The ELEON Authoring Environment

ELEON is a metadata authoring environment designed allowing domain experts to author RDF annotations without requiring knowledge of Sevod to the level of manually editing the RDF statements. We foresee two use cases in the context of SemaGrow metadata:

- When new data sources are added to the federation, or the schema employed in a data source is updated, or several instance-level changes (dataset size, entity URI patterns) have accumulated, the data provider can use ELEON to manually provide or update data source annotations; and
- For metadata that is automatically maintained by the SemaGrow Stack, so that it can be inspected and possibly corrected.

In general, ELEON can be used by human operators to inspect and edit more complex data source annotations than what can be represented by the simple configuration files described above. In fact, ELEON visualizes the complete information in SemaGrow histograms, including information can be extremely difficult to check and validate or correct by human operators.

3.2.1 Installation and execution

ELEON is published as open source software under the GPLv2 license and is developed on a public Bitbucket repository. To checkout the current version, ELEON 3.0, please issue:

 # git clone https://bitbucket.org/semagrow/eleon.git && cd eleon && git fetch && git checkout tags/v3-0

To checkout the latest version on the ELEON 3.X branch please issue:

 # git clone https://bitbucket.org/semagrow/eleon.git && cd eleon &&git fetch && git checkout eleon3

Besides its dependencies on Java libraries, Java JDK 7 and Apache Ant are also needed in order to compile ELEON.

The binary distribution of ELEON only depends on Java SE Runtime Environment 7; all libraries used by ELEON are included in the ELEON distribution. The binary distribution is available at http://iit.demokritos.gr/~eleon

After extracting the binary distribution, the following directory structure is created:

./	Contains eleon.jar and eleon.sh
./resources/	
./resources/schemas/	Contains schemas and vocabularies that ELEON needs for its operation.
./resources/testcases/	Contains testcases for different ELEON domains and applications.
./resources/testcases/dataset/	Contains testcases for the Semagrow *data source annotation* use cases.

This structure *should not* be removed or relocated. The tool is executed by issuing:

 # java -jar eleon.jar

from the ./ directory. In order to have an executable that does not depend on execution path, please modify eleon.sh to set the ELEON installation path. In order to run the tool, *Java Runtime Environment version 7* must be installed. JRE 7 can be installed as follows on Debian-based systems:

 # apt-get install openjdk-7-jre

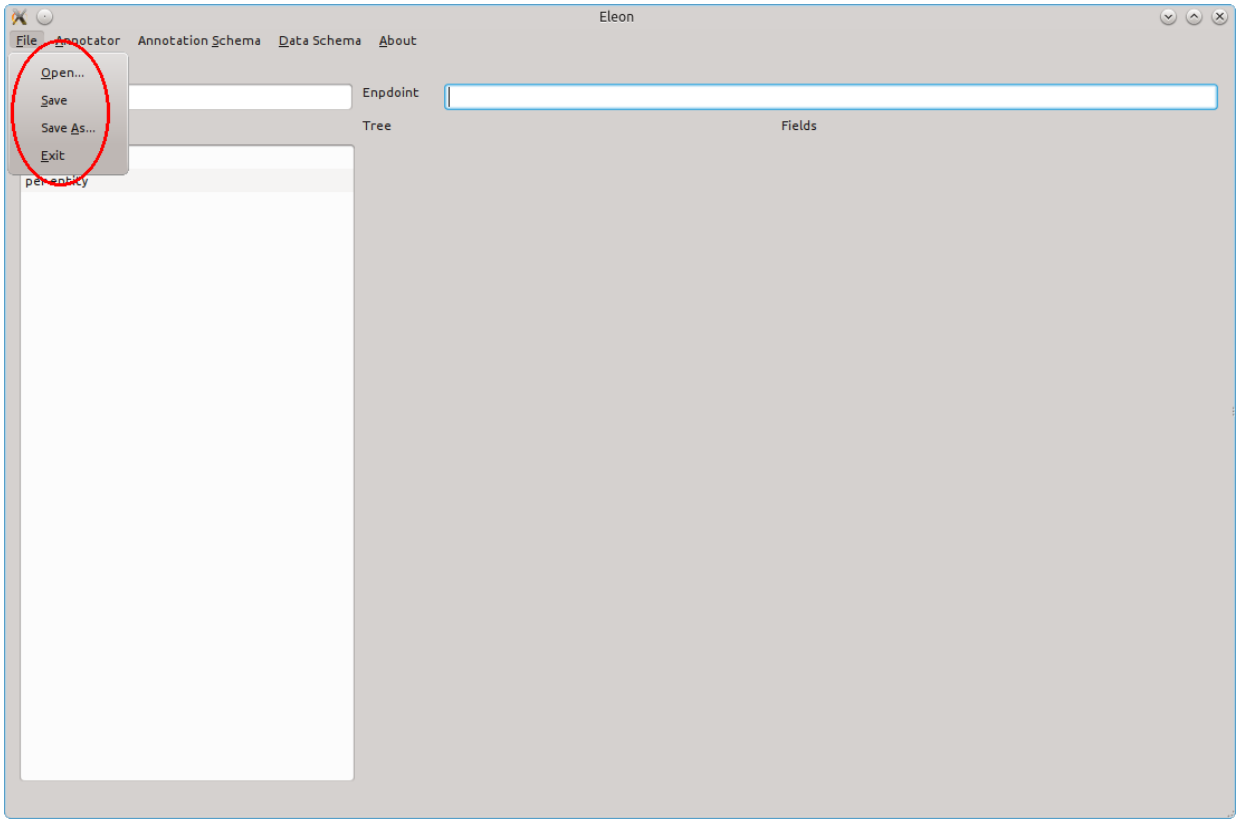

Figure 6: Loading and saving annotations

3.2.2 Starting an annotation session

To resume a previously saved annotation session the user has to load the annotations from thefile selection dialog at the *File->Open* menu (Figure 6). When the file is loaded the previous annotations will appear in the UI. After the annotation process has finished the user can save the annotations from the *File->Save* and *File->SaveAs*menus.*File->Save* overwrites the file most recently opened or saved-as. When ELEON terminates, all changes since the last save are lost.

After loading an annotations repository, users must authenticate in order to gain write access to a particular segment of the overall annotations repository. Users have read-only access to the complete repository, but write access only to annotations owned by their own username. The level of security provided depends on the backend, and in the case of RDF files discussed here, it amounts to simply declaring the user's identity in the *Author* menu by either selecting an existing user name or adding a new one (Figure 7). User identities persist in the annotations repository so user names from previous sessions are available after a repository has been loaded.

When starting from scratch, the annotation schema must be selected. The *Annotation Schema* menu offers the list of schemas known to ELEON and allows selecting *exactly one* of the available options (Figure 8). At the moment, the *VoID* and *Sevod* schemas are available; *Sevod* is the pre-set default.

Then the user must select the schemas that will be known to ELEON as the possible schemas used in the data sources being annotated. This can be done from the *Data Schema*menu. The annotator can select one or more schemas from the menu or load a new one by selecting an OWL, RDF/XML, or TTL file from the local filesystem (Figure 9). The schemas that are pre-installed and do not need loading are in the resources/schemas/directory of the distribution and should not be removed or relocated.

It should be noted that the data schemas selected in this menu to not refer to the data schema used in any particular dataset; they are the schemas relevant to this session as a whole and the schemas among which the user may select the schema that any specific dataset follows.

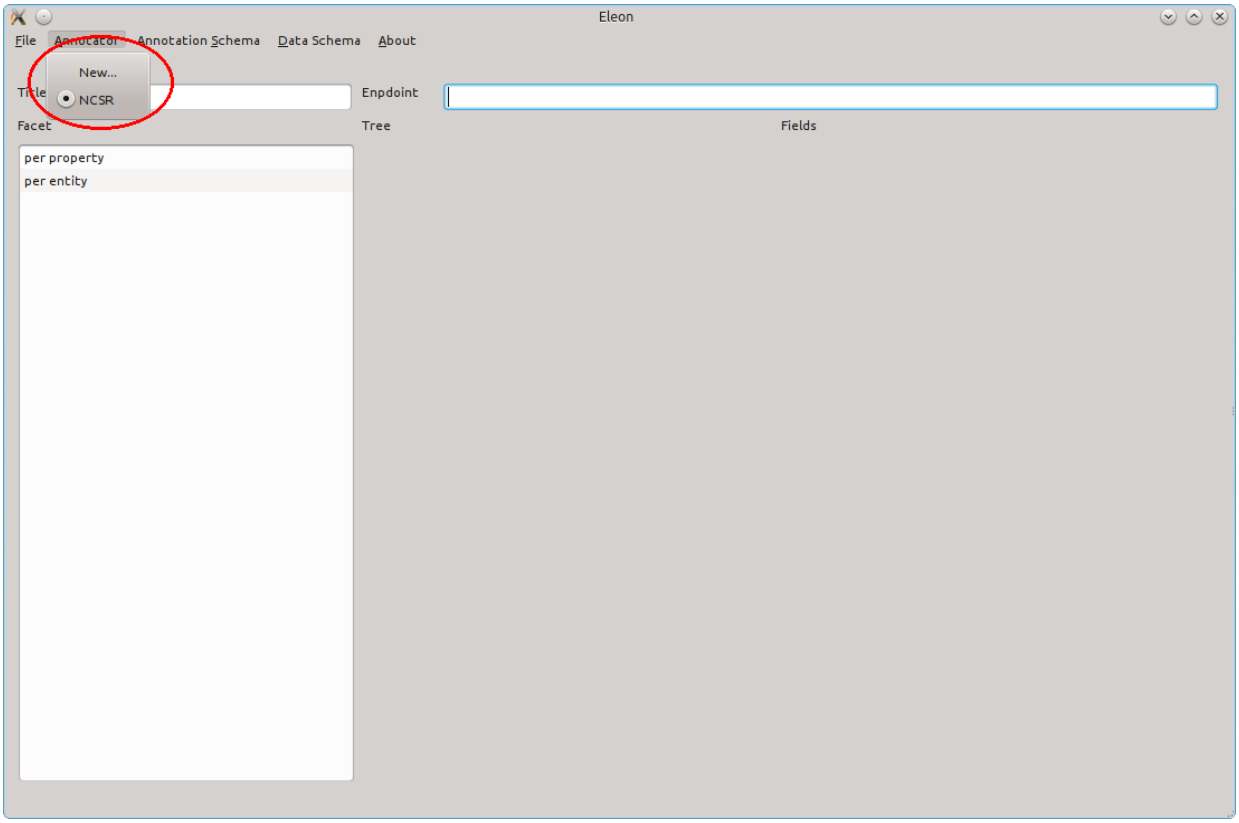

Figure 7: Login menu

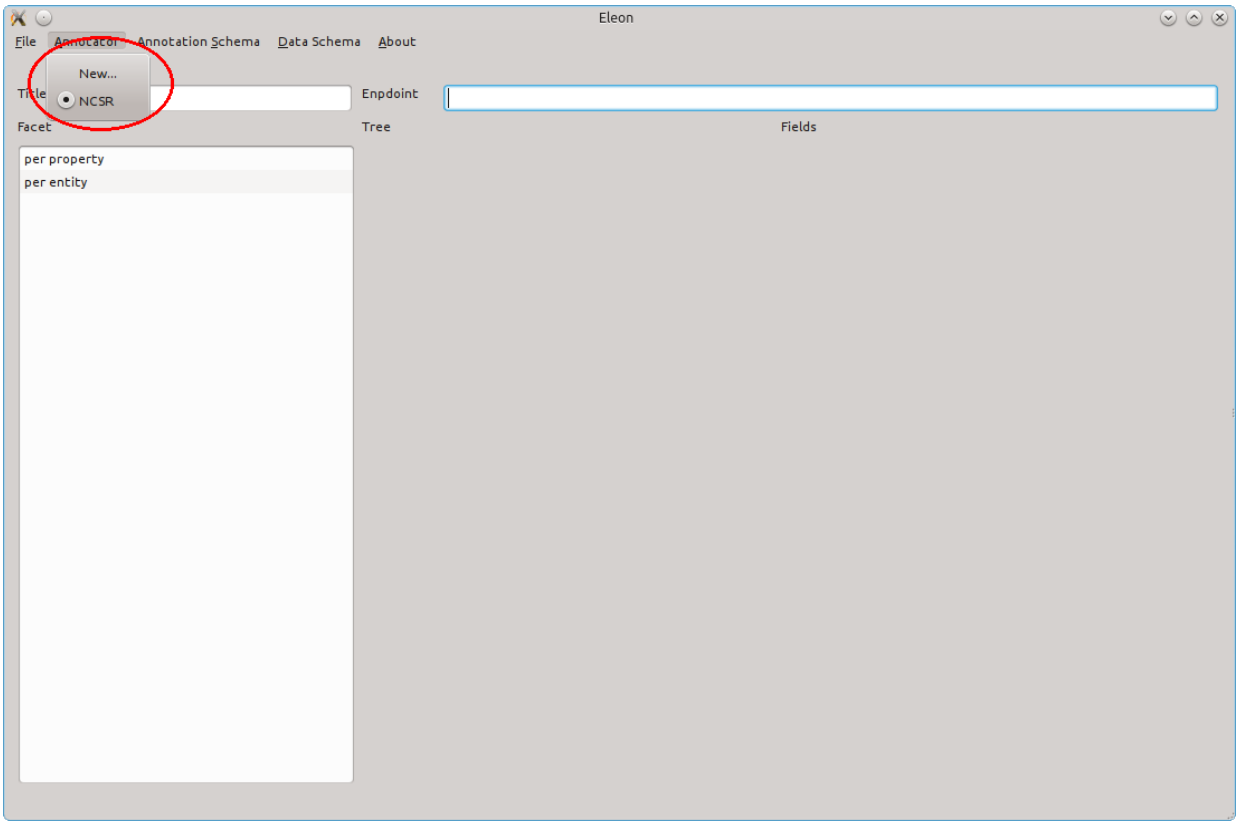

Figure 8: Annotation Schema

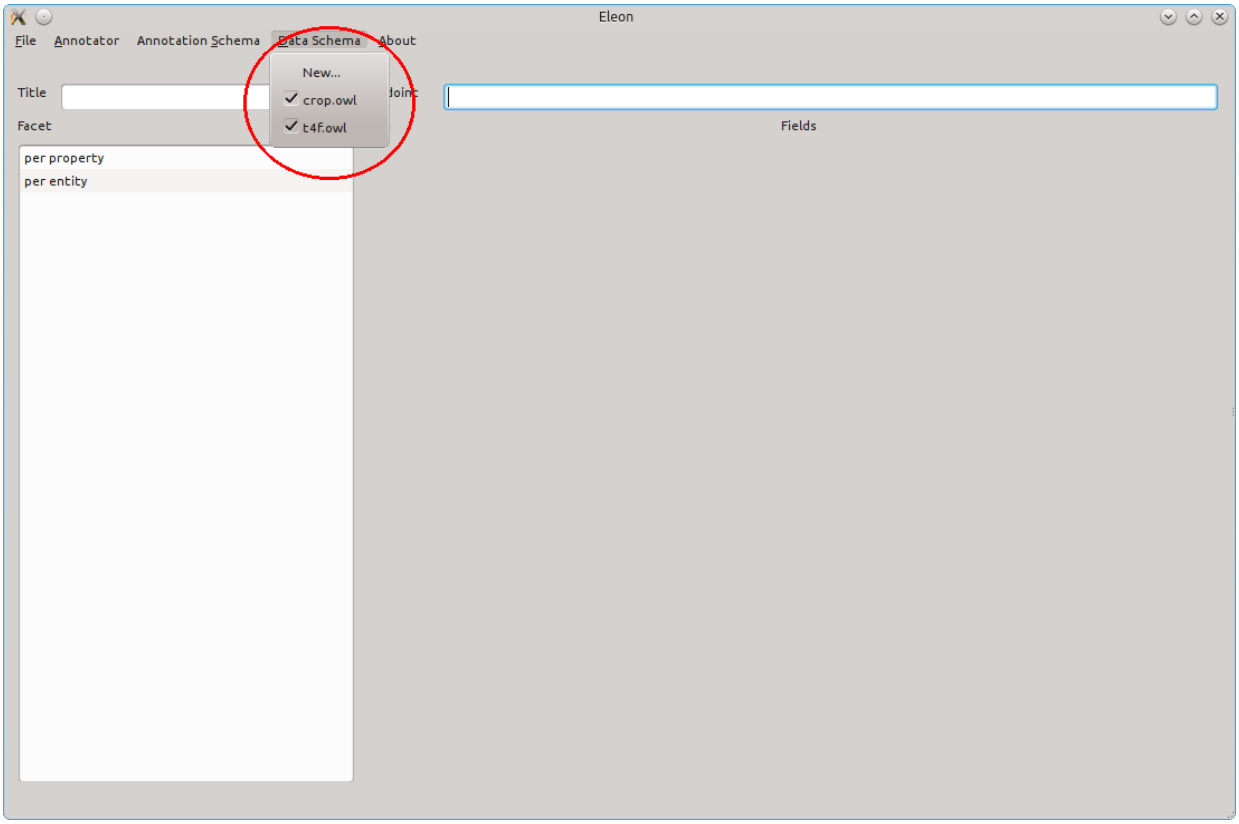

Figure 9: Data Schema menu

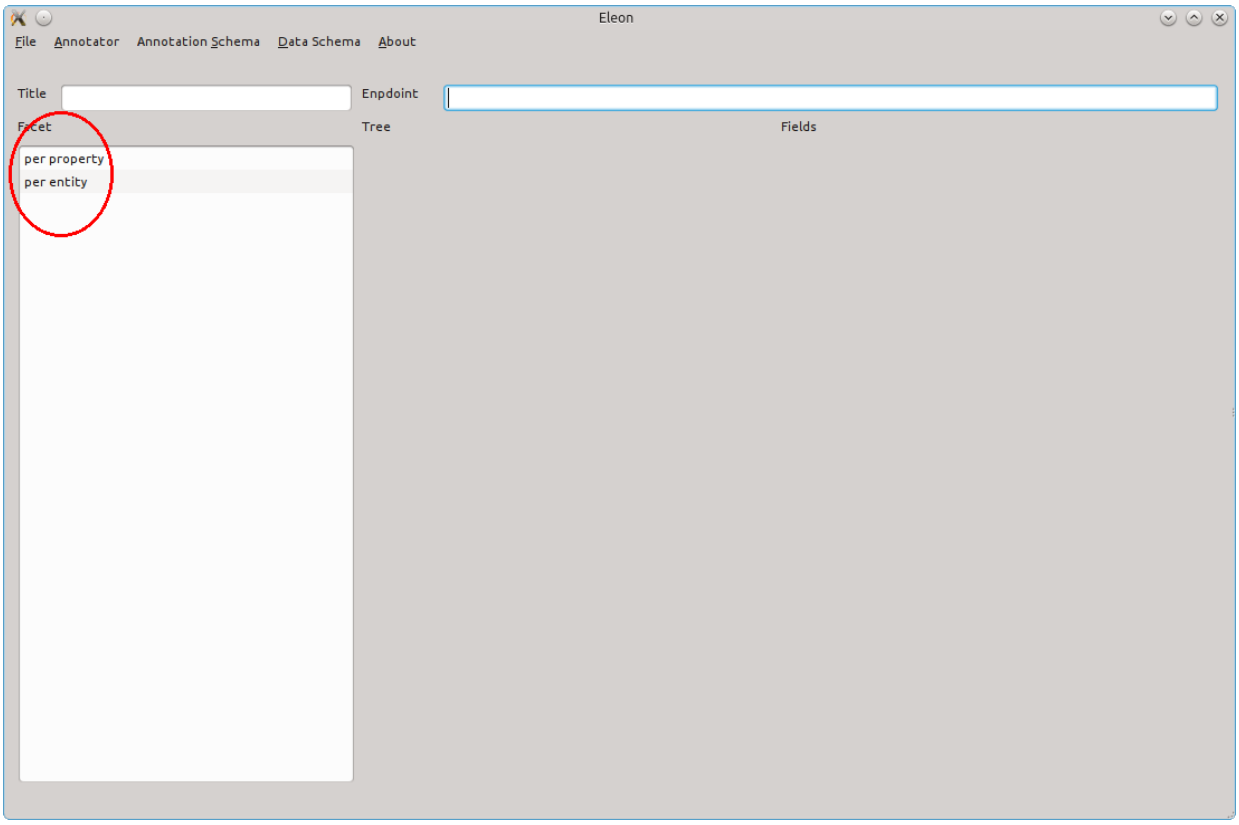

Figure 10: Dataset browsing facets

3.2.3 Faceted browsing

The annotator then can start the annotation process. To do so he has to select one of the available facets from the *Facet* panel (Figure 10). A tree will be drawn next to the facet list containing a node named root. This node cannot be deleted.

The annotation process now depends from the type of facet the user selected. First we will describe the process for the "per property" facet and then for the "per entity" facet.

3.2.3.1 Per Property annotation

Before clicking on the "per property" facet an Annotator (Figure 7) and at least one data schema (Figure 9) have to be selected.

To begin the annotator has to define a new dataset to annotate. To do so he has to right-click the "root" node of the tree and the click on the "Insert dataset label" menu item (Figure 11). A new dialog will appear in which the user must put the name of the dataset (Figure 12). The dataset with the given name will then appear in the tree under the root node (Figure 13). To annotate this dataset the user must click on the node. A table will appear next to the tree containing in one column the properties available for annotation and the in the second column a text field to insert values for those properties (Figure 14).

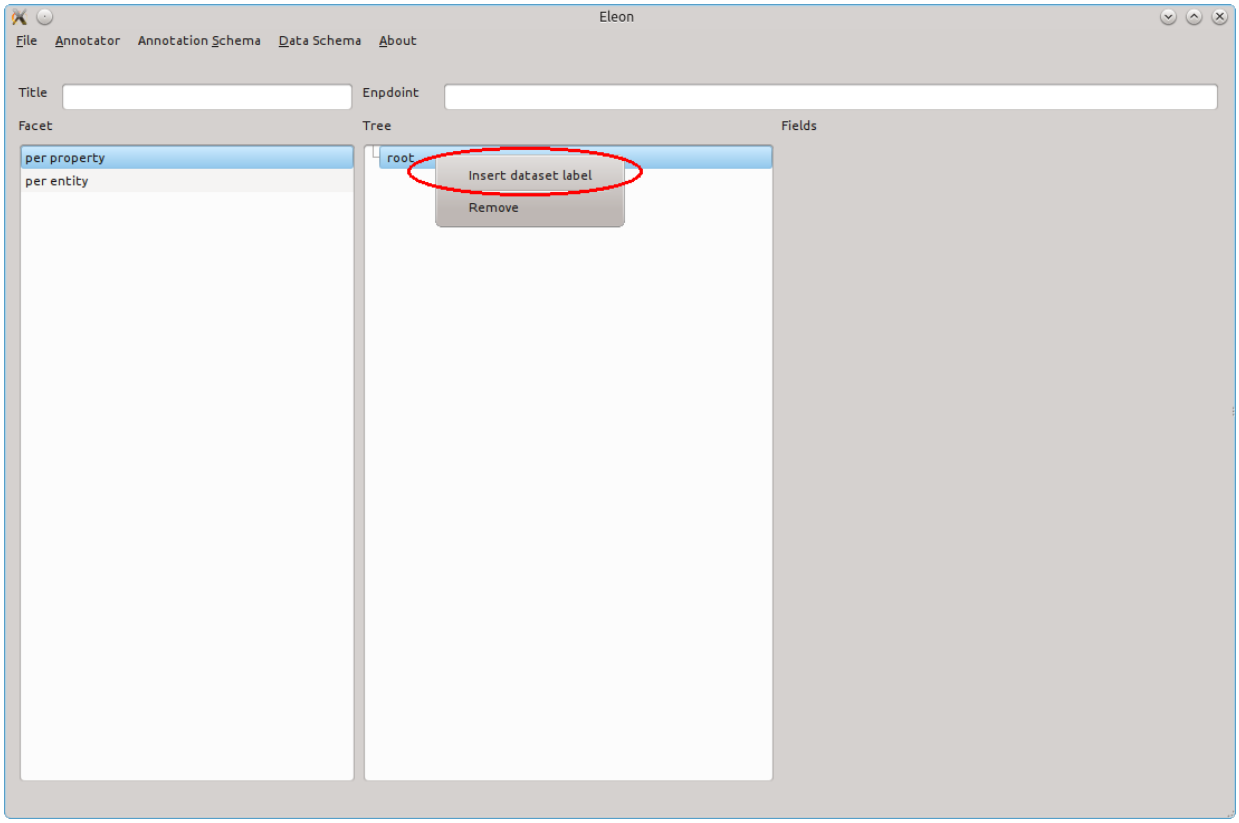

Figure 11: Insert dataset label

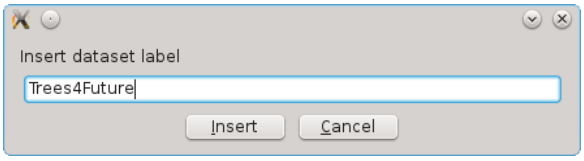

Figure 12: Dataset label

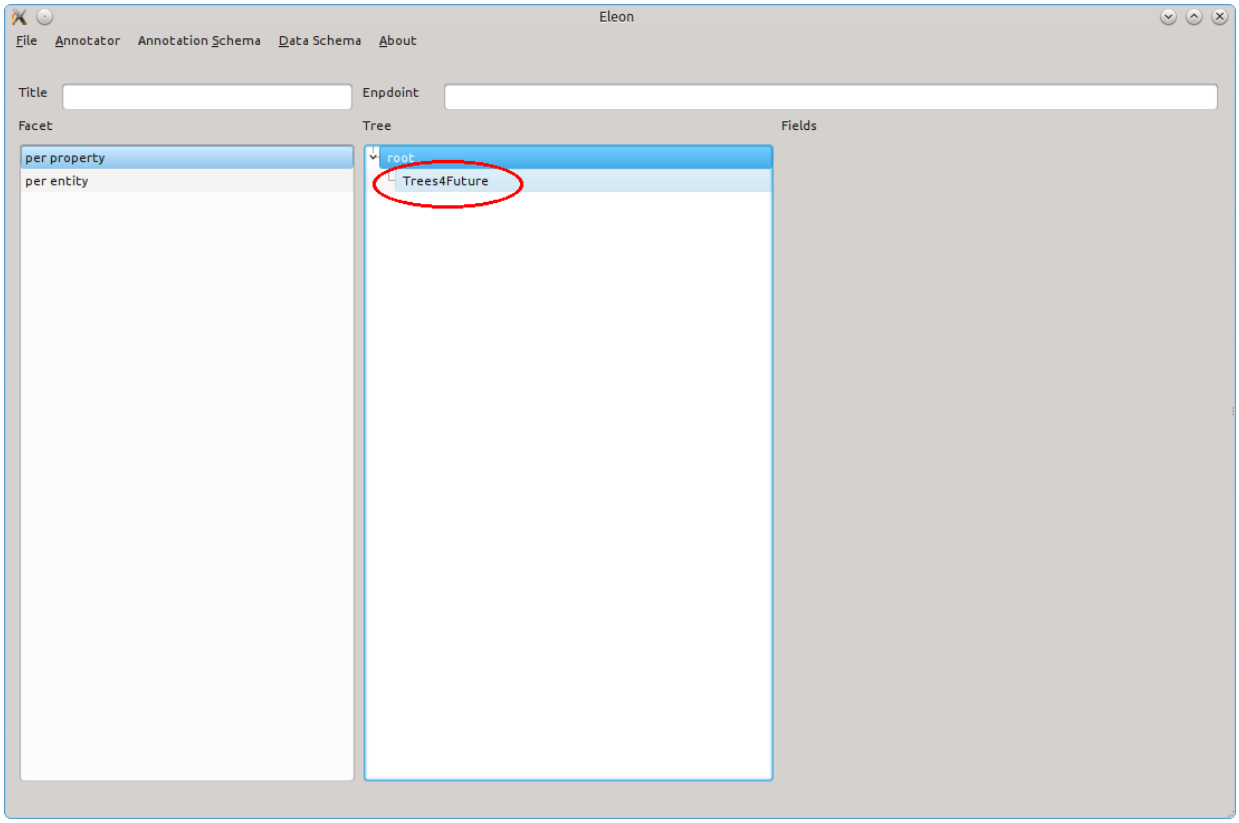

Figure 13: New dataset inserted

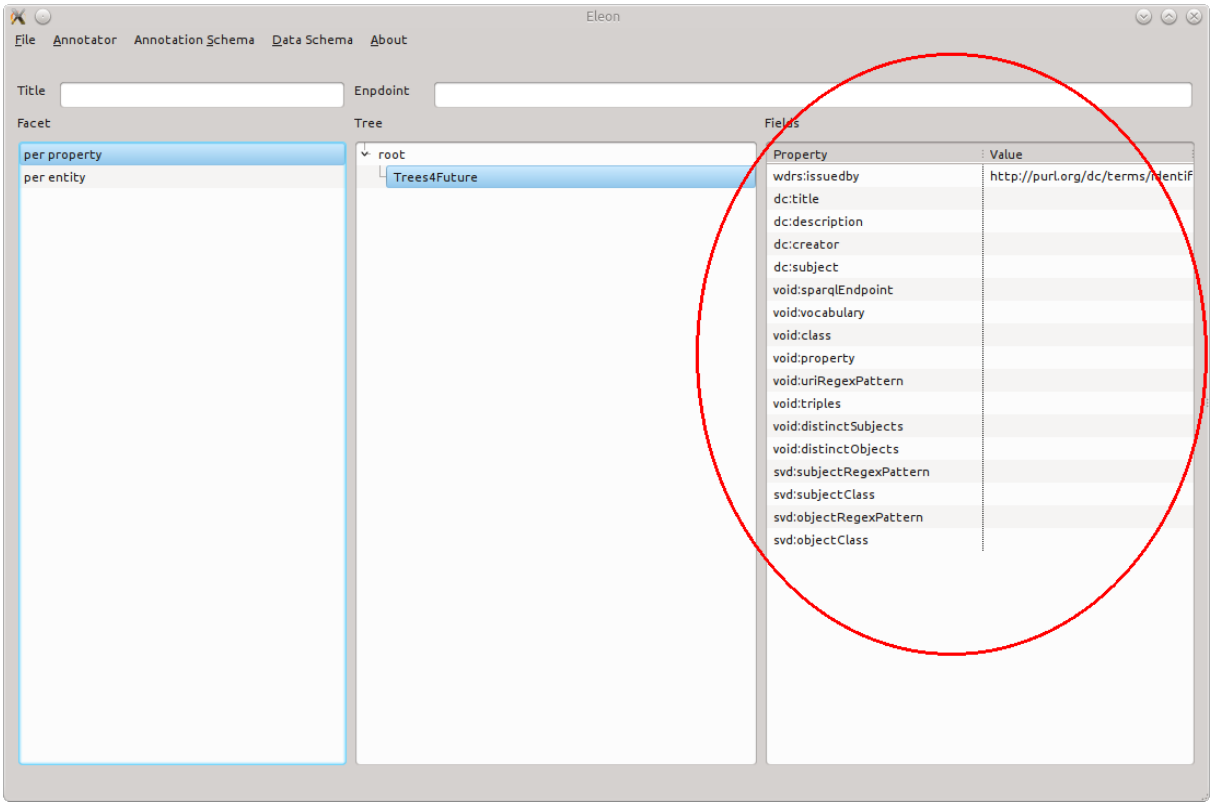

Figure 14: Property editor

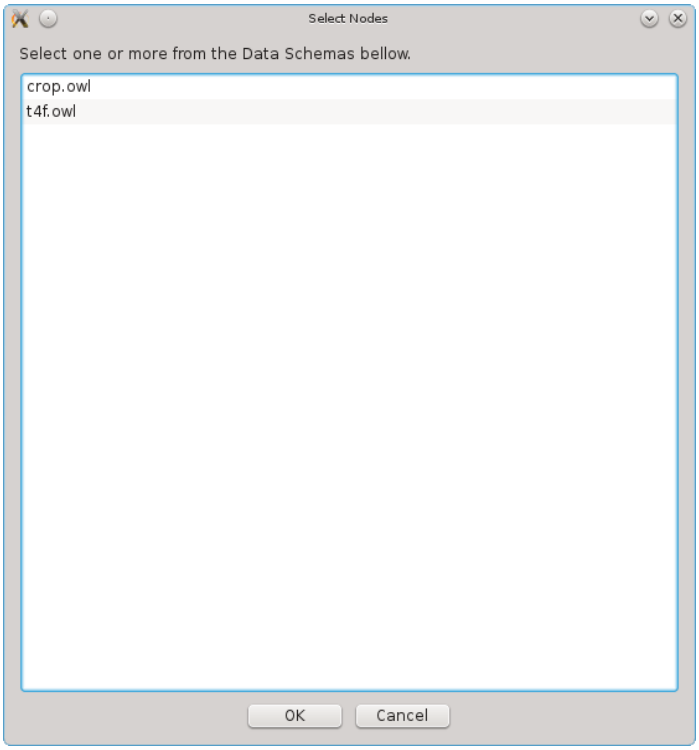

Figure 15: Schema selection

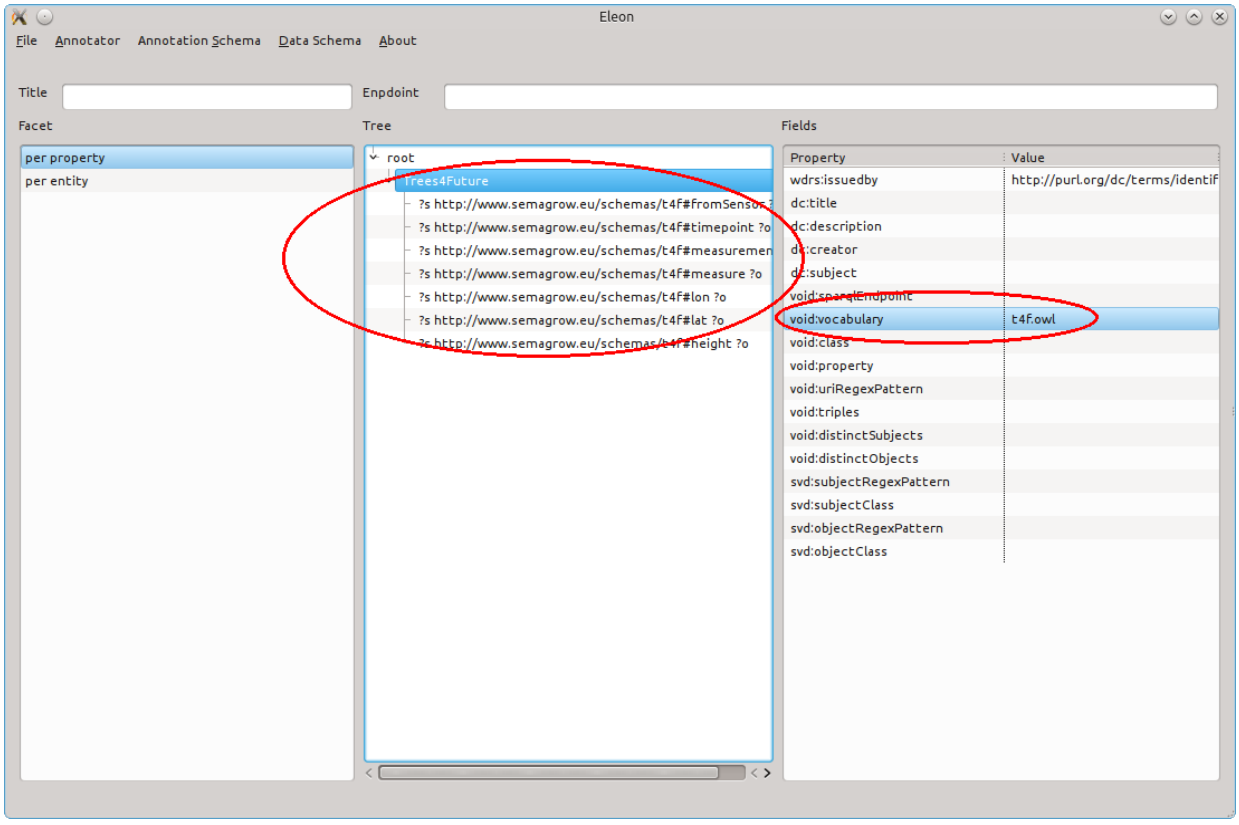

Figure 16: Schema added

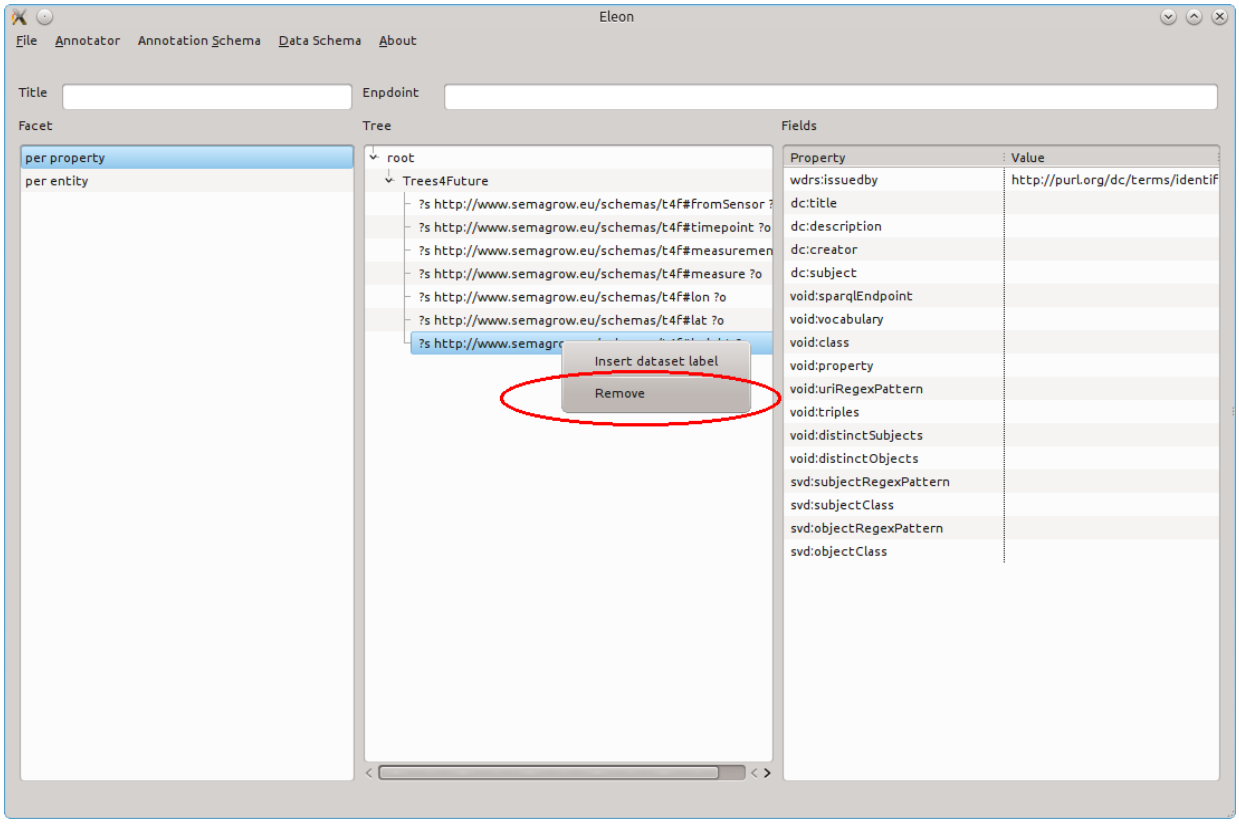

Figure 17: Remove from tree

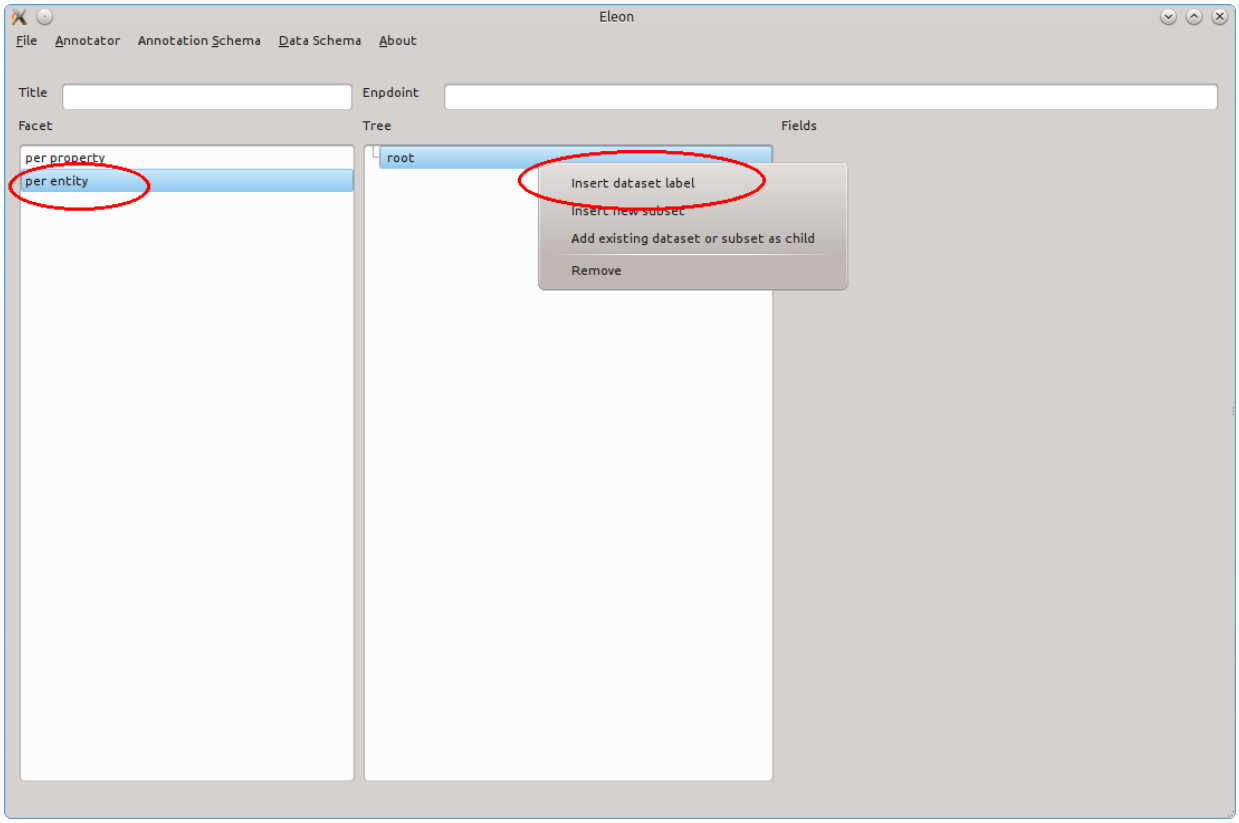

Figure 18: Insert dataset

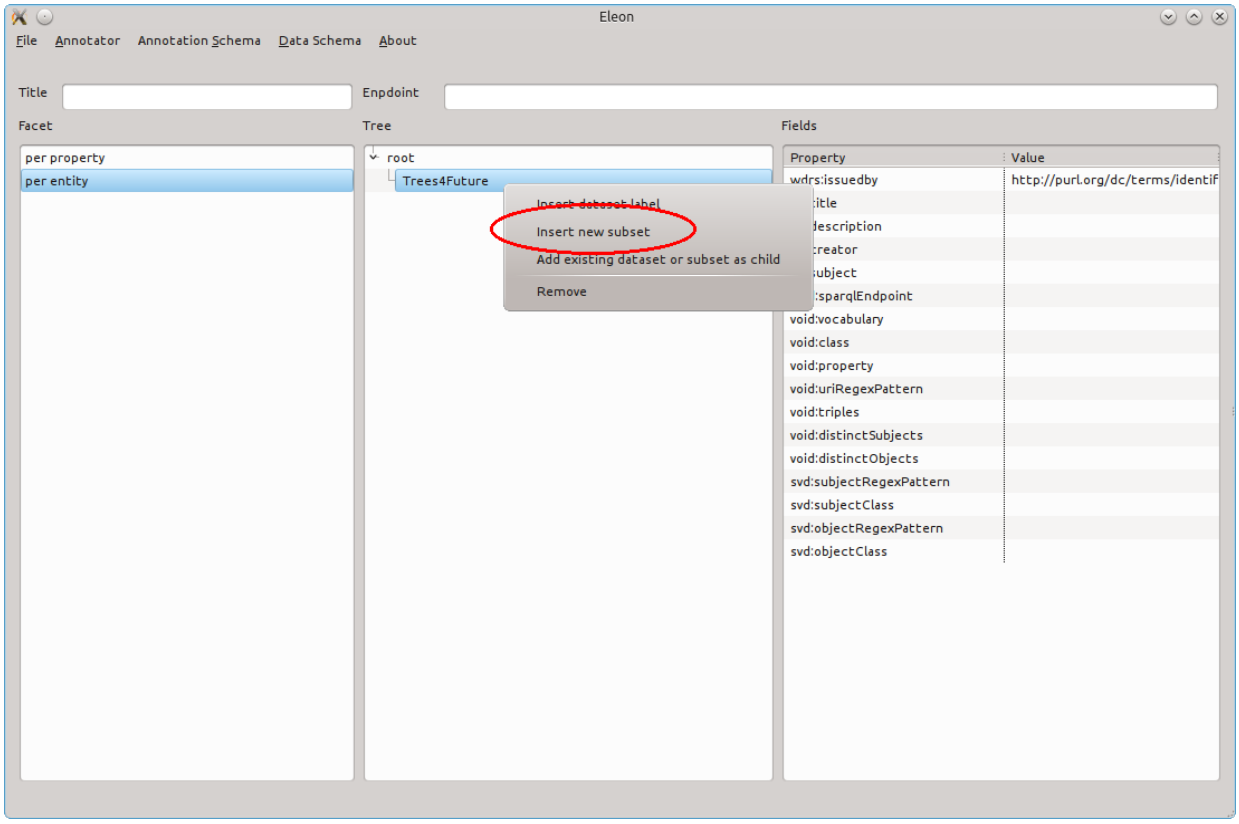

Figure 19: Insert new subset

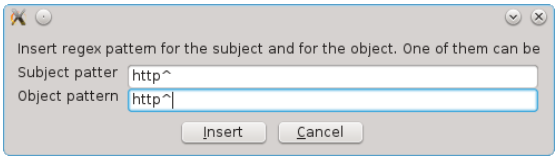

Figure 20: Subject and Object pattern definition

The user then must click on the “void:vocabulary” field. A new window will appear containing a list with all the available schemas, asselected from the *Data Schema* menu. The user has to select one or more from the list and click the “OK” button (Figure 15). After that the properties from the selected schemas will appear under the selected data source (Figure 16). The annotator can also delete a node from the tree by right-clicking of the node and the click on the “Remove” button. The node and all its children will be removed from the tree (Figure 17).

The annotator can then insert values for the rest of the properties by editing the fields in the property table (Figure 14). Some properties, depending on the Annotation Schema selected (Figure 8), are auto-filled by the tool and cannot be edited by the annotator.

3.2.4 Per Entity annotation

The annotation process for the “per entity” facet is similar to that of the “per property” facet. Again the user has to click on the facet, a tree will appear next to the facet’s list containing a root node and the user can insert a dataset to annotate (Figure 18). Then the annotator can insert a new subset by right-clicking on a dataset and selecting “Insert new Dataset” from the menu (Figure 19). A new dialog will appear in which the annotator has to input the regex pattern for the subjects and/or the objects (Figure 20) of this subset. The new subset will then appear as child of the dataset with label “<subject_pattern> ?p <object_pattern>” (Figure 21).

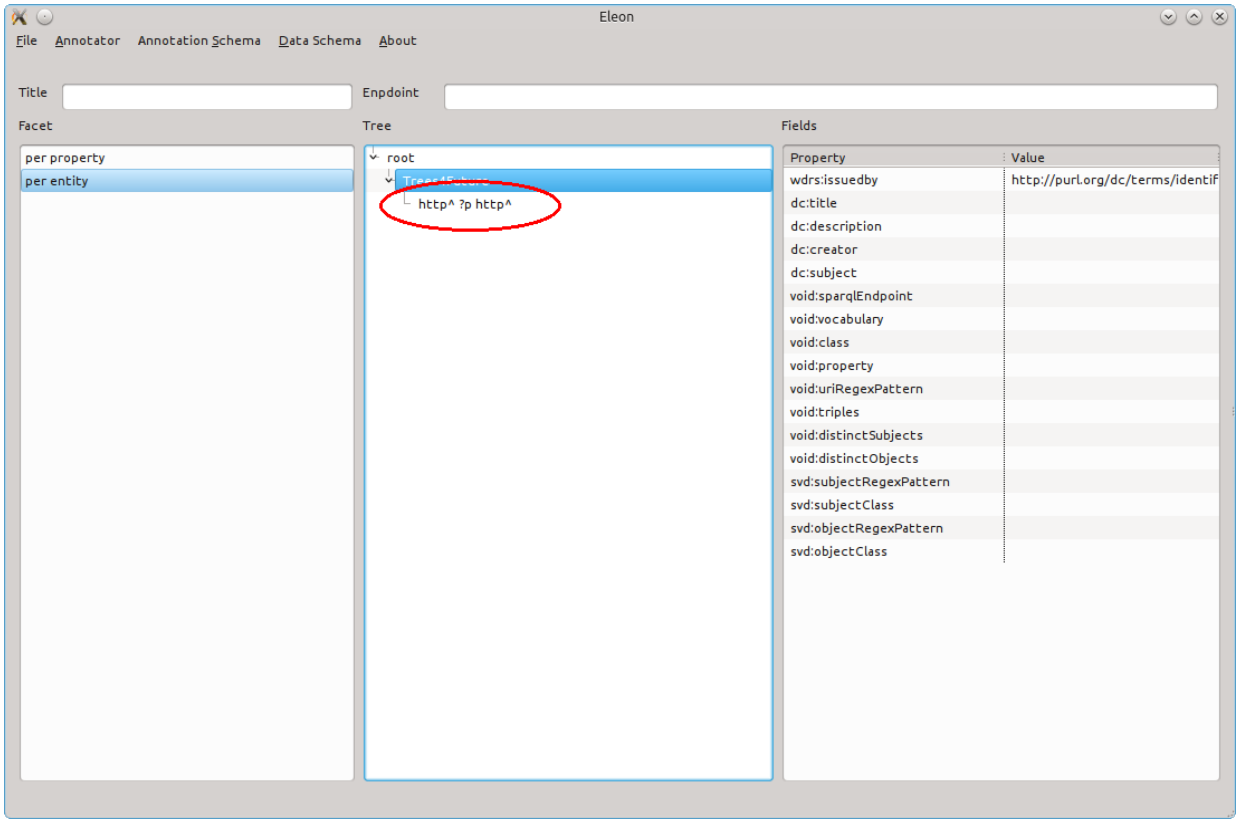

Figure 21: Subset added

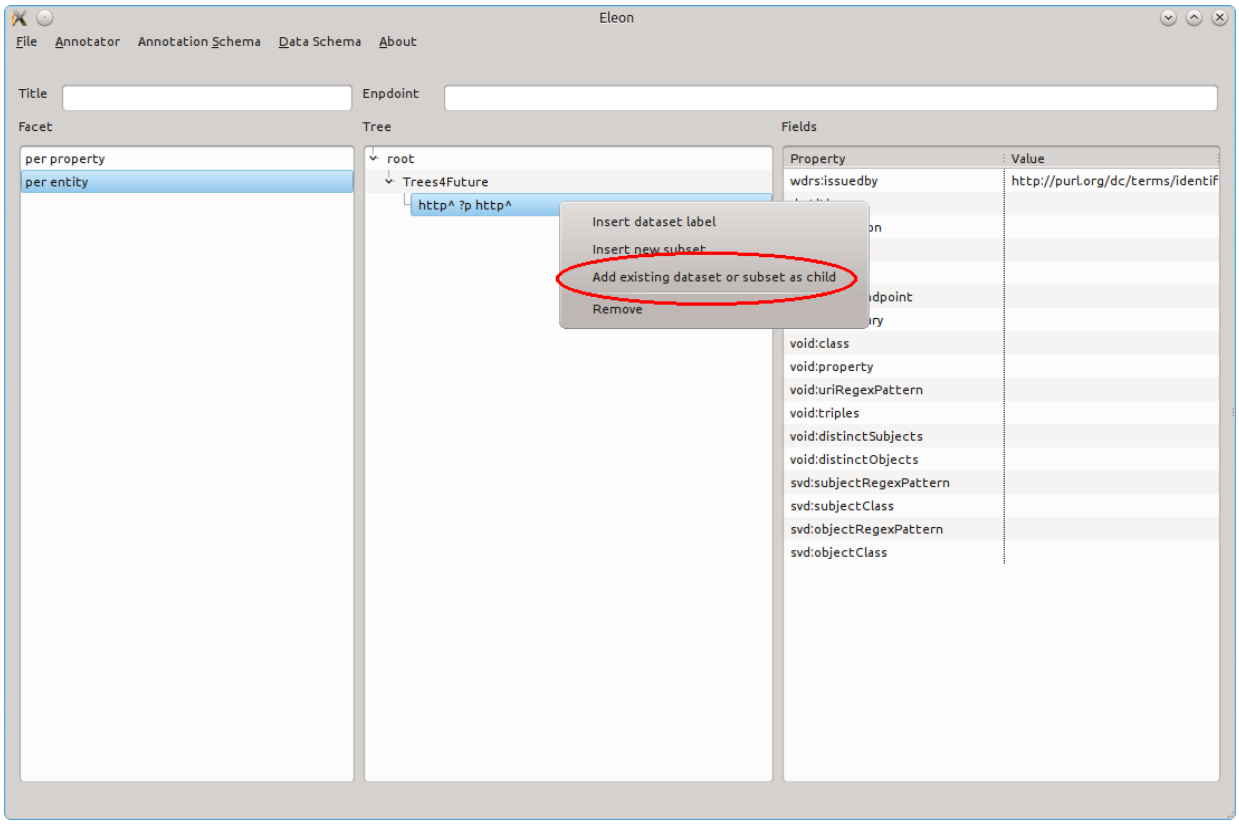

Figure 22: Add existing dataset or subset as child

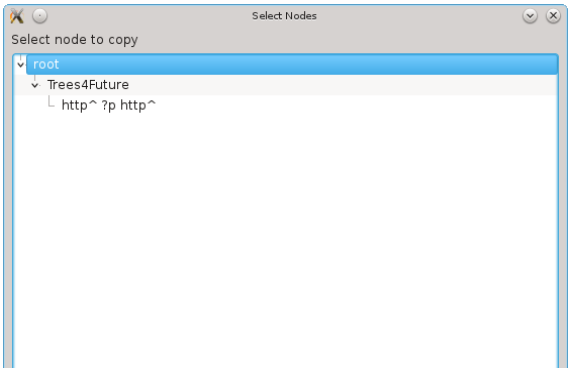

Figure 23: Multiple inheritance

The annotator can also select a previously created dataset or subset to insert by right-clicking on any tree node and selecting “Insert existing dataset or subset as child” (Figure 22).

A dialog will appear and the annotator can choose one dataset or subset to be copied under the selected node (Figure 23). The annotation cannot copy a subset under the root node. Only datasets are permitted under that node. The annotator can add values to the properties of each dataset or subset by using the property editor (Figure 14).

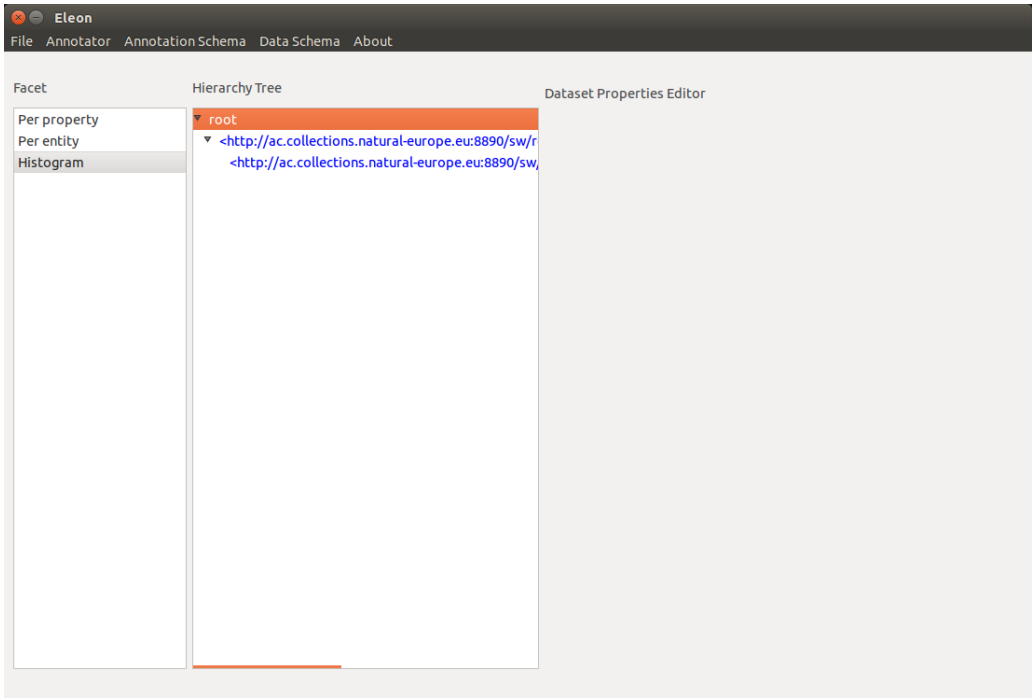

Figure 24: Histogram Tree Structure

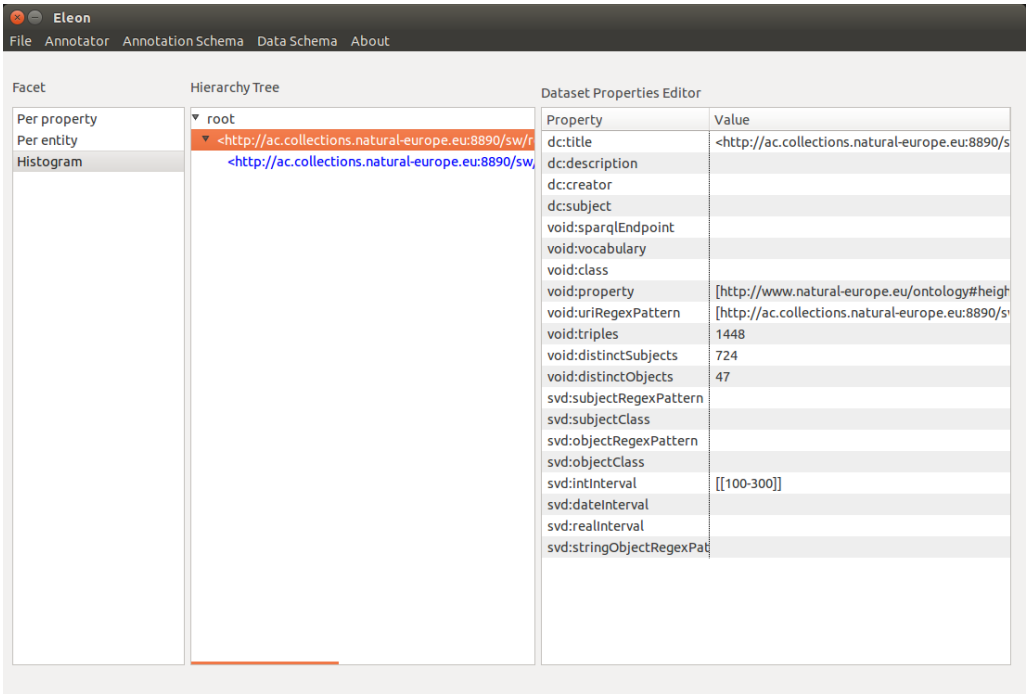

Figure 25: Histogram Properties Editor

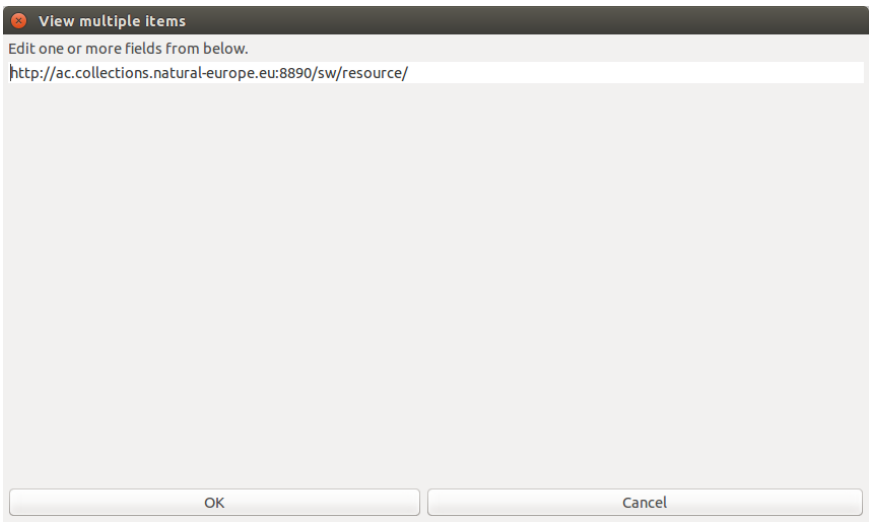

Figure 26: Editing URI regular expressions

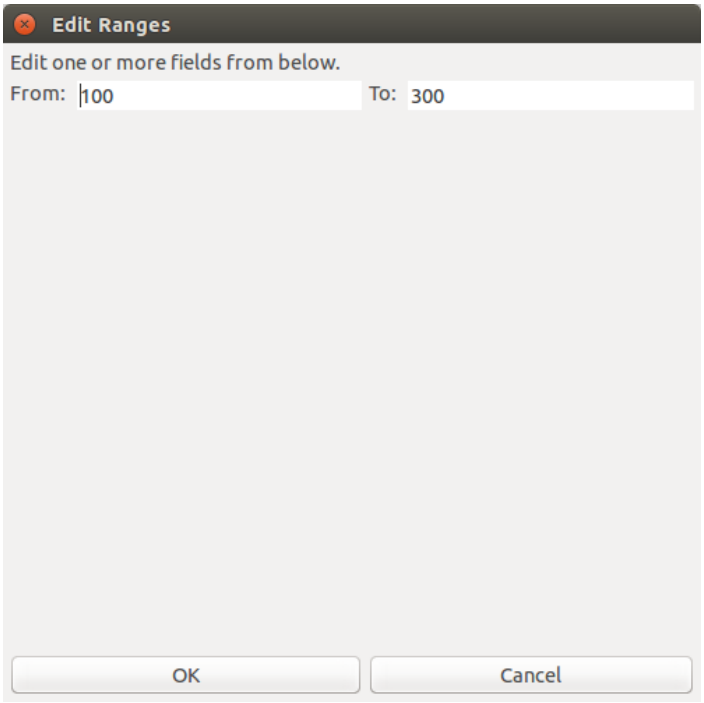

Figure 27: Editing numerical rages

3.2.5 Histogram annotation

The annotation process for the “histogram” facet is similar to that of the “per property” and “per entity” facet. Again the user has to click on the facet, a tree will appear next to the facet’s list containing a root node and the user can insert a dataset to annotate (Figure 24). Make sure to select an Annotator and a Schema in order to be able to act on the facet.

The difference from the other facets is that the dataset is automatically generated by a histogram so the user must load an existing dataset by clicking *File->Open* and selecting a file to load.

After the successful loading of the histogram and selection of the proper facet, ELEON will visualize it based on a tree structure. The automatically generated nodes of the tree are colored in blue and can be edited by any user of the system (Figure 25). A user can click on a node and the available meta-data will appear on a right table, available for editing (Figure 26). Nodes that have been edited by another user of the system can’t be edited by anyone else, based on ELEON authorization policy, and will appear in red color. The user that is currently logged-in will see the nodes, edited by him, with green color. Proper validity checks exist to help user insert valid information. A user can edit URI ranges (Figure 26) as well as numerical ranges (Figure 27).