Project ICT 287534
Start: 2011-09-01
Duration: 36 months
Co-funded by the European Commission within the 7th Framework Programme

**SEMANCO Semantic Tools for Carbon Reduction in Urban Planning**

SEMANCO

# Deliverable 4.3
# User interfaces for domain experts interacting with SEIF

**Revision:** 6
**Due date:** 2013-02-29 (m18)
**Submission date:** 2013-04-30
**Lead contractor:** HAS

| Dissemination level | | |
|---|---|---|
| PU | Public | X |
| PP | Restricted to other program participants (including the Commission Services) | |
| RE | Restricted to a group specified by the consortium (including the Commission Services) | |
| CO | Confidential, only for members of the consortium (including the Commission Services) | |

| Deliverable Administration & Summary | | | | | |
|---|---|---|---|---|---|
| **No & name** | **D4.3 User interfaces for domain experts interacting with SEIF** | | | | |
| **Status** | Final | **Due** | m18 | **Date** | 2012-04-30 |
| **Author(s)** | Michael Wolters, German Nemirovski (HAS), Joan Pleguezuelos, Álvaro Sicilia (FUNITEC) | | | | |
| **Editor** | German Nemirovski (HAS) | | | | |
| **DoW** | User interfaces for domain experts interacting with SEIF: The interfaces working with the different SEIF components accessible in Internet. User interface screencast showing the functionalities and examples of energy data interrelations. | | | | |
| **Comments** | | | | | |

| Document history | | | |
|---|---|---|---|
| **V** | **Date** | **Author** | **Description** |
| 1 | 2013-01-29 | Michael Wolters (HAS),<br>German Nemiovski (HAS) | First draft |
| 2 | 2013-04-16 | Joan Pleguezuelos (FUNITEC) | Semantic data explorer contributions |
| 3 | 2013-04-17 | Álvaro Sicilia (FUNITEC) | Introduction and general editing |
| 4 | 2013-04-18 | German Nemirovski (HAS) | Last changes and consistency check for references and number of figures |
| 5 | 2013-04-22 | Álvaro Sicilia (FUNITEC) | Final editing. This version is sent to internal reviewers: Martin Carpenter (UoT), Xavi Cipriano (CIMNE). |
| 6 | 2012-04-27 | Michael Crilly (UoT) | Final proof-reading |

# Table of Contents

# EXECUTIVE SUMMARY

Deliverable 4.3 *User interfaces for domain experts interacting with SEIF*, developed within Work Package 4 *Semantic energy information framework*, summarizes the work done and the results achieved in Task 4.3 *User interfaces for knowledge representation*, whose goal is the development of front-end tools to facilitate ontology designers and domain experts the operations with data.

Deliverable 4.3 contributes to the project with:

- An ontology **editor** which fulfils requirements established through application of the ontology design methodology – explained in Deliverable 4.2 *Semantic Energy Model*–, concerning the quality of the ontology code and increasing efficiency of the coding process by automated generation of complex code constructions, such as *DL-Lite*$_A$ axioms. The editor has been developed as a tool for cooperative ontology design, involving ontology designers and domain experts, such as building engineers and energy consultants. Therefore, it has to fulfil particular requirements concerning usability. In this regard the editor offers a set of advanced visualisation features such as simultaneous representation of ontology from two different perspectives: on the fly inferring and graphical visualization of relations between concepts, widgets for customisation of presentation and easy access to editor's functions, e.g. search, versioning or annotation.
- A **semantic data explorer** environment to enable domain experts to explore the data exposed by the Semantic Energy Information Framework (SEIF). The environments make use of the ontology to support the users in the query formulation. The data explorer is composed of three tools: the ***search***, the ***relate***, and the ***query***. Since the SEIF provides data which is already integrated, this tool enables users to find and navigate through the relations between different data sources.

  Although these three tools (the ***search***, the ***relate***, and the ***query***) have different goals and tasks to fulfil, they share two common environmental components, i.e. the administration module and the query engine. The former enables experts to configure the SPARQL endpoints by selecting query optimization settings. The query engine uses the input of the user, in form of keywords, to create a SPARQL code dynamically and to retrieve data from corresponding SPARQL endpoints. The generated code depends on the settings selected using the administration module.

The report is structured in the following sections:

1. **Introduction**: Purpose of the deliverable, contributions of partners, and relationships between the work undertaken in this task with other work packages.
2. **Ontology Editor**: Description of the background, initial situation and requirements formulated for the ontology editor; its features and technical details of implementation including software architecture and a brief description of libraries used in the implementation.
3. **Data Explorer**: Description of the background, initial situation and requirements formulated for the data explorer; its features and technical details of implementation. This section also describes the administration module and query engine. The functionality of the tools is illustrated with examples and screenshots.

4. **Conclusions**: Contributions of the user interfaces to the project development, and results, particularly as a set of instruments facilitating domain experts to influence processes of data integration and management.

# 1 INTRODUCTION

## 1.1 Purpose and target group

The purpose of this deliverable is to report about the work done in Task 4.3 *User interfaces for knowledge representation*.

Two environments have been developed in this task:

- an **ontology editor** which hides the complexity of coding ontologies in *DL-Lite*$_A$ formalism which is a requirement established in the ontology design methodology explained in D4.2. The editor has been used in the TBox coding task of the ontology design methodology to create the SEMANCO ontology. The editor has been used by ontology engineers and domain experts working collaboratively.
- a **semantic data explorer** environment to enable domain experts to explore the data exposed by the Semantic Energy Information Framework (SEIF). The environments make use of the ontology to support the users in the query formulation. The semantic data explorer is composed of three tools: the ***search***, the ***relate***, and the ***query*** tools. Since the SEIF provides data which is already integrated, this tool enables users to find and navigate through the relations between different data sources.

Ontology designers and technicians are the main target groups of this document. However the environments developed in this task is also of interest of domain experts since they are an important part of the users group targeted by the tools described in this document.

## 1.2 Contribution of partners

The work carried out in Task 4.2 was led by HAS and FUNITEC. The ontology editor has been implemented by HAS. The semantic data explorer has been developed by FUNITEC. The report has been written by both HAS and FUNITEC.

## 1.3 Relations to other activities in the project

The environments created in the Task 4.3 give support to the project development. Specifically the ontology editor has been used in the TBox coding sub-task of the ontology design methodology. The ontology editor is an important output of the project since it has helped to save time and reduce code errors through automation of the coding process for more than three thousand axioms.

In addition, the semantic data explorer environment gives support to domain experts to navigate through the data provided by the three cases studies. It is a neutral and generic interface to explore the data relations which complements the platform tools focusing on specific use cases with the final goal of reducing $CO_2$ emissions. The relationships with previous and future tasks and deliverables is summarized in Figure 1.

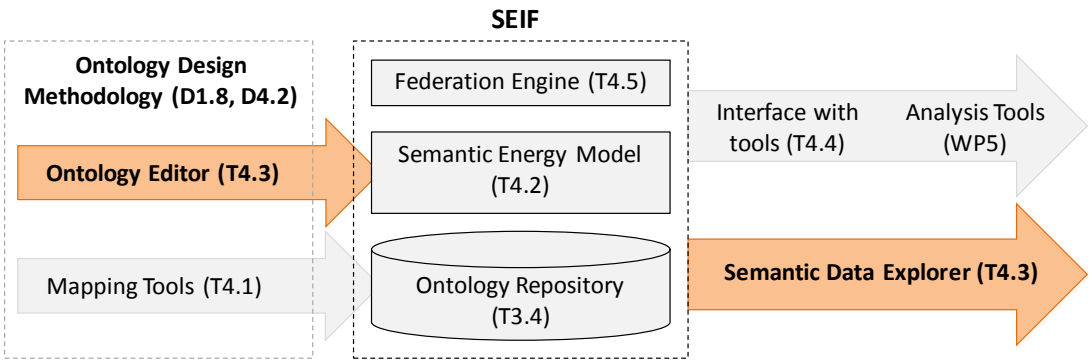


*Figure 1. Relationships with other tasks and deliverables*

# 2 ONTOLOGY EDITOR

In the early stage of the SEMANCO project two comparative studies for ontology design tools provided in Khondoker & Mueller (2010) and in Kapoor & Sharma (2010) have been analysed. In addition to these, selected tools such as Protégé, WebODE and OntoEdit can be found in Knublauch et al. (2004), Arpírez et al. (2001), and Sure et al. (2002) have been tested. We identified that tools for ontology design evaluated in the early project stage were missing instruments for user input validation/control with respect to rules and restrictions determined by a particular description logic like $\mathcal{AL}$, $\mathcal{SHOIN}$, $\mathcal{SHIQ}$, $\mathcal{EL}$ or by an OWL profile (OWL 2 QL, OWL 2 EL and OWL 2 RL). The demand on more intensive support of the ontology designers is just natural, especially if domain experts are entrusted with this task.

Providing an analogy with software engineering, on the fly code control is an essential state of the art feature implemented in numerous integrated development environments (IDEs), such as *Eclipse*[1]. Similar to software designers who are notified immediately after having typed incorrect code, ontology designers should be notified if a specification of an ontology element is not aligning with the definition of the selected profile or DL logic. For example, if such specification uses a constructor that is not available in the selected profile or logic. Alternatively an ontology designer should be provided strongly restricted GUIs that would not able to generate incorrect code, i.e. the GUI available for a designer must not enable specification of constructers which are not defined in the selected profile or logic. Furthermore, to our mind, an ontology design environment in the 21th century should also provide on-the-fly validation of semantics being specified by the user's input.

These were the primary considerations which moved the SEMANCO team to develop a new tool for the ontology design. Besides these considerations, additional requirements were formulated with regard to the specific goal of the ontology design, i.e. integration of energy consumption and $CO_2$ emissions related data, and concerning the design methodology illustrated in Deliverable 4.2.

## 2.1 Requirements on the Ontology Editor

At the start of the project, these two basic requirements for the ontology editor were specified:

- It would support code generation addressing the needs and restrictions arisen in the context of data integration
- It would have a high usability and authoring efficiency for fast processing of large ontologies

In the following chapter we will show which features of the editor fulfil these requirements.

## 2.2 Supporting code generation in the context of data integrations

As stated in Deliverable 4.2 the DL-Lite$_A$ formalism was selected for the ontology coding because of its special features which facilitate data integration. Furthermore, formalisms of the DL-Lite family to which DL-Lite$_A$ belongs to, serve as a basis for the OWL QL profile of OWL 2 designed for the purpose of data accessing/management.

The basic properties of DL-Lite$_A$ are specified in Poggi at al. (2008), with only some being

[1] http://www.eclipse.org/

referred to within this report. This combination of properties is aiming to achieve a compromise between maximum expressivity of the specification language and the optimisation of time performance when processing conjunctive queries. Such queries are basically ABox queries typically aiming at collecting a set of properties related to a particular group of individuals. In fact these queries are one of the most important techniques for data integration, and thus in turn, the primary technological challenge of the SEMANCO project.

As stated in Nemirovski et al. (2012) the most important features of DL-Lite$_A$ are that: 1) domain and range of properties can be specified only for functional data properties; and 2) definition of an object property connecting two OWL classes with each other has to be modelled by means of axioms and not by specifying property's domain and range. For example, two following axioms in DL notation use subsumption (⊑), existence quantification (∃) and inversion ( ) to express that the class Horizontal_Superior_Enclosure relates to the class Roof via the hasRoof property.

$$\exists \text{hasRoof} \sqsubseteq \text{Horizontal\_Superior\_Enclosure} \quad (1)$$
$$\exists \text{hasRoof} \sqsubseteq \text{Roof}$$

In OWL the same is specified as follows:

```
<owl:Restriction>
 <rdfs:subClassOf rdf:resource="http://www.ontologyportal.org/SUMO.owl#Roof" />
 <owl:onProperty>
  <rdf:Description>
       <owl:inverseOf rdf:resource="http://www.semanco-          (2)
       project.eu/2012/5/SEMANCO.owl#hasRoof" />
  </rdf:Description>
 </owl:onProperty>
 <owl:someValuesFrom rdf:resource="http://www.w3.org/2002/07/owl#Thing" />
</owl:Restriction>
```

and

```
<owl:Restriction>
    <rdfs:subClassOf rdf:resource="http://www.semanco-           (3)
     project.eu/2012/5/SEMANCO.owl#Horizontal_Superior_Enclosure" />
    <owl:onProperty rdf:resource="http://www.semanco-
     project.eu/2012/5/SEMANCO.owl#hasRoof" />
 <owl:someValuesFrom rdf:resource="http://www.w3.org/2002/07/owl#Thing" />
</owl:Restriction>
```

Although domains and ranges of properties are not explicitly specified in the code, if an ontology specification is valid, they can be inferred by reasoner software to be then visualised and viewed by the user.

In this context the SEMANCO ontology editor should control generation of code that has to be strictly aligned with the limitations and requirements of DL-Lite$_A$. In other words, the editor should limit the resulting OWL code to the elements available in DL-Lite$_A$. As long as the number of these elements is small such limitation can be achieved through restricting of editing options for the user to a set of GUI widgets, like context menu options. Each of these options generates, deletes or modifies an OWL code corresponding to a particular DL-Lite$_A$ element. Therefore the SEMANCO ontology editor, as most of currently available ontology editors, does not support free code editing in terms of XML code.

In DL-Lite$_A$ ontologies, the code similar to one shown above is always required to specify how two concepts are connected to each other. For conventional ontology editors , for

example; Protégé, the specification of axioms provides probably the biggest difficulty when editing a DL-Lite$_A$ conform ontology. When coding these axioms in Protégé, actions have to be carried out in three difference windows: class description, object properties hierarchy and general class axioms. Errors are inevitable when the number of such connections increases in this manner. In the SEMANCO ontology editor such axiomatically specified relations are generated by a click on a context menu option (Figure 2).

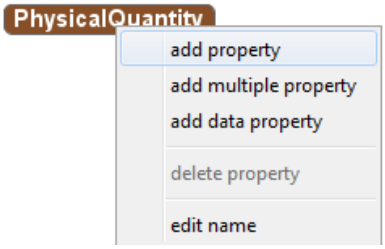


*Figure 2. Context menu selection in SEMANCO ontology Editor leads to generation of complex codes*

## 2.3 High usability and authoring efficiency

To improve usability and authoring efficiency, the following features were implemented in the ontology editor:

- Supporting of collaborative design process carried out by ontology engineers in cooperation with domain experts.
- Using on the fly inferring for visualisation of an ontology graph and interactive graph navigation.

### 2.3.1 Support of collaborative design process

In particular the SEMANCO ontology editor supports collaborative design by:

- Separating tasks by subdividing complex tasks into simpler ones, i.e. separation of specification processes for i) subsumptions; ii) other (than subsumption) relations between concepts; iii) concepts annotation using metadata
- Providing mechanisms for version, users and change management

The editor developed for these purposes is shown in Figure 3. The editor presents the ontology in two different ways:

- As a taxonomy, i.e. a hierarchy of concepts subsuming each other (Figure 3, left)
- As a hierarchy of non-subsumption relationships (Figure 3, right).

Therefore, the ontology coding is partitioned in two sub-processes. Domain experts basically connect concepts by non-subsumption, i.e. has-relationships working in the right part of the window. New concepts emerging during this process are initially subsumed by the concept *Suspense*. Later, in an independent sub-process, all sub-concepts of *Suspense* are relocated to an appropriate position within the taxonomy. This sub-process is carried out as a mutual task by ontology engineers and domain experts. Such process separation simplifies thinking and helps to avoid confusing relations between concepts.

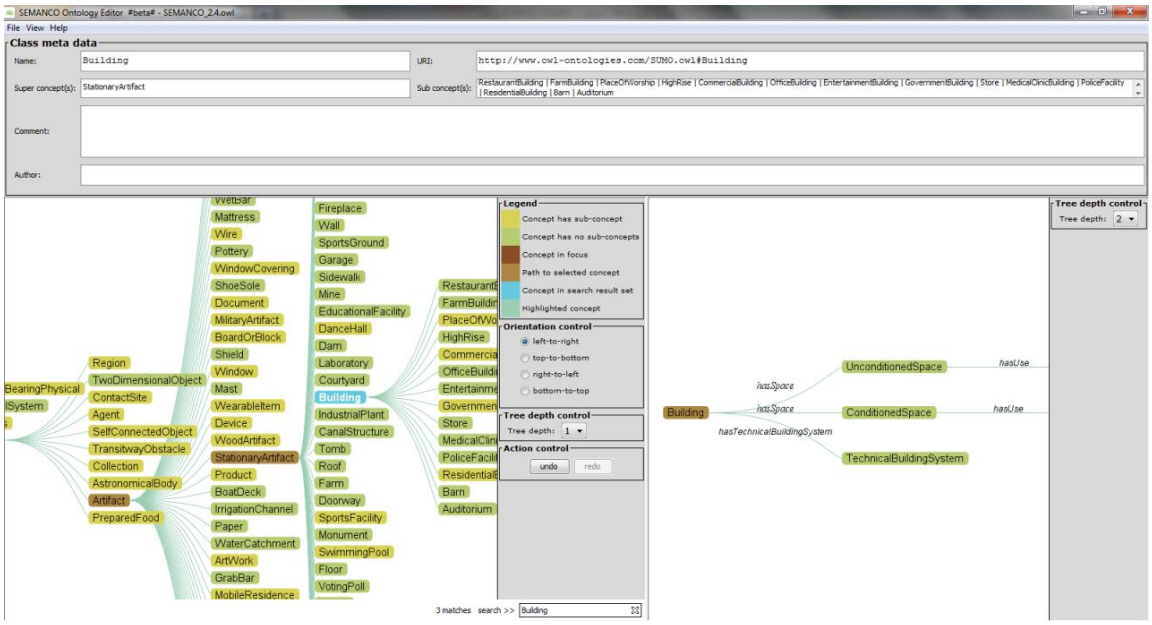


*Figure 3. SEMANCO ontology editor*

### 2.3.2 Visualisation of an ontology graph and interactive graph navigation

It is important not only to enable the user to navigate through a visual representation of the taxonomy of concepts, but also to provide a self evident graphic representation of non-taxonomic (non-subsumption) relations between concepts (as one specified in the code example (2) and (3)).

For this purpose the SEMANCO ontology editor is able to infer on the fly relations between ontology elements, to present inferred relations as a graph and to show parts of this graph in a user friendly form (Figure 4).

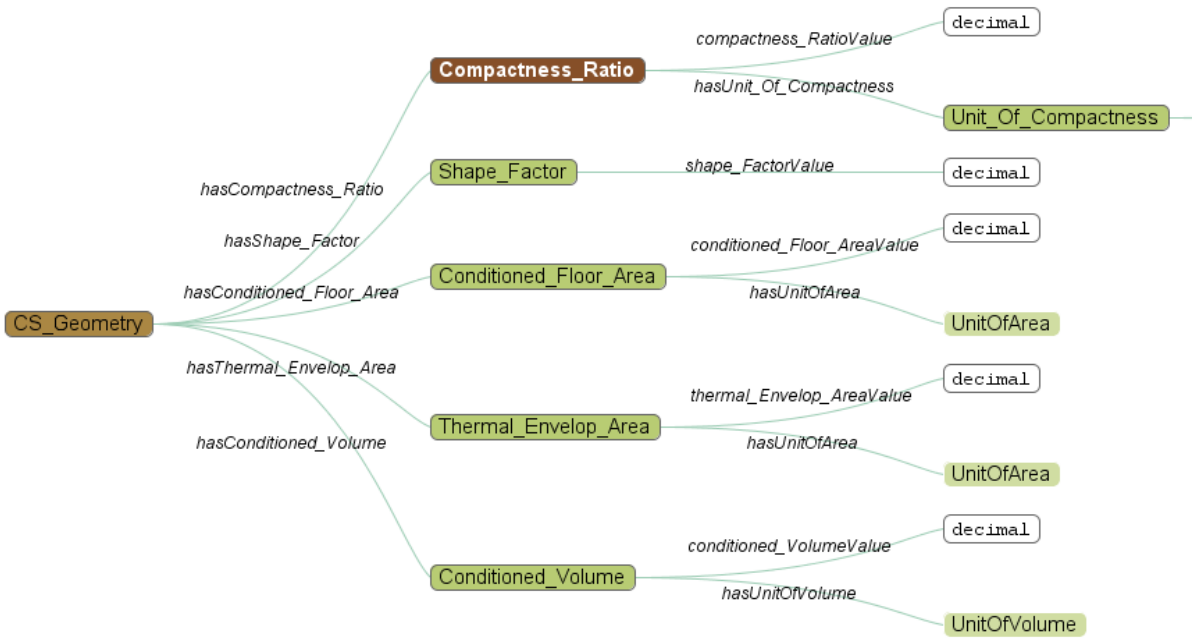


*Figure 4. Inferred relations as a graph*

The SEMANCO ontology editor offers to the user, the following instruments for navigation and definition of the presentation settings.

- Coloured presentation of concept nodes (the legend in Figure 5 is shown to the user permanently)
- Mouse click navigation: After a concept is clicked with the left mouse button it moves to the focus of the window (node CS_Geometry in Figure 4) and new relations edges originating from this node and from its sub nodes become visible (the node Compactness_Ratio in Figure above). After a double click a concept becomes a root of the shown part of the graph (the node CS_Geometry in Figure 4)
- Zoom function for enlarging/collapsing parts of the graph
- Widget control for the depth of the relationship tree (Figure 5)
- Selection of graph direction
- Widget control for the forward/backward navigation along the navigation history list (Figure 5)

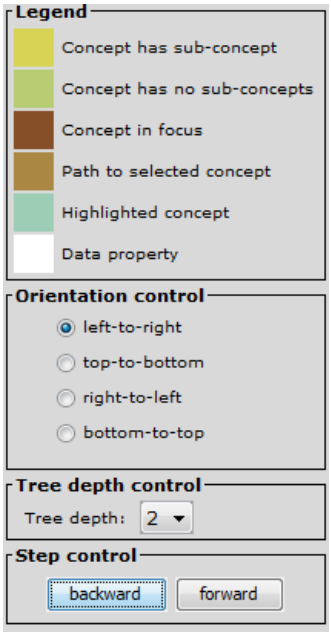


*Figure 5. Control panel of the SEMANCO ontology editor*

## 2.4 Technical characteristics of the implementation

The SEMANCO ontology editor is implemented in Java Version 7 as a *Swing* graphical user interface. Three major frameworks are integrated: *Prefuse*, *Apache Jena* and *JDOM*. The underlying software architecture conforms to the model-view-controller (MVC) design pattern.

The current version of the SEMANCO Ontology-Editor is 1.4.9 and consists of 23 single files with more than 5200 lines of Java code.

### 2.4.1 Model–view–controller (MVC)

A Model View Controller (MVC) software architecture separates processes and components responsible for data management and business logic from those responsible for user interface

and therefore offers a good reusability. A graphical user interface (*view*) is implemented in the *MainGUI* class. A couple of small, additional views amend this GUI, e.g. the *SearchGUI* and the *AddPropertyDialog*. The *MainController* class takes all user input, like mouse clicks, and sends them as request to the *MainModel* class implementing the business logic of the software.

### 2.4.2 Prefuse Visualisation Toolkit

The *Prefuse*[2] visualisation toolkit was created for visualising all kinds of data and offering interactive control over them. The selection of this framework is due, amongst other reasons like reliability, code quality and support offered online, by the fact that a Java implementation of this framework is already available. The framework expands the Java 2D graphics library and can easily be integrated into existing Java Swing software. All sorts of graphs, trees and tables can be displayed statically and dynamically, used for querying and integrated with databases.

For the SEMANCO ontology editor, the tree functionalities of *Prefuse* are used to display both views representing different sorts of relations between concepts (Figure 4): the subsumption view (left part of the editor window, showing only taxonomy of subsumptions) and the non-subsumption view (right part of the editor window, representing all non-subsumption relations).

### 2.4.3 Apache Jena

The Apache *Jena* framework[3] has been developed for creating Semantic Web applications using the programming language Java. It offers a lot of tools and libraries for reading, processing and writing RDF data in XML format; for dealing with OWL ontologies and for rule-based reasoning.

The *Jena* framework is most notably used in the *MainModel* class of the SEMANCO ontology editor. With the help of *Jena* the SEMANCO ontology is imported and transformed into a Jena ontology model (*OntModel*). The ontology Model is then used as basis for all actions with the ontology, i.e. to read or create *Object-Properties*, *Classes*, *Annotations* and so on.

### 2.4.4 JDOM

*JDOM* is an open source Java-based document object model for XML (http://www.jdom.org/). In the SEMANCO ontology editor *JDOM* is used to manage (access and modify) files written in the Web Ontology Language (OWL), whereby the OWL is derivate of the XML, defined by means of a set of name spaces. The classes *SaveToOWL* and *XMLWriter* offer all the required functions to add, change or delete ontology elements.

### 2.4.5 Ontology import functionalities

The on-the-fly inferring described in Section 2.3.2 is carried out using several class methods. Within the *MainModel* class, the method *detectObjectPropertiesRelations()* is responsible for processing the semantic information stored in several hundred different OWL restriction tags as (2) and (3) which implement *DL-Lite*$_A$ axioms as (1) shown in Section 2.2. This method analyses such axioms for each OWL object property and infers *domain* and *range* definitions for these properties. It supplements and extends the functionality offered by the basic *Jena* methods such as *getDomain()* and *getRange()* of class *OntProperty*.

[2] *Prefuse* is licensed under the terms of a BSD license; for more information please go to http://prefuse.org/ .
[3] For more information see http://jena.apache.org/.

The *OWLTreeConverter* class is responsible for preparing the visualisation of data. Whereas the creation of the *prefuse* tree for the subsumption view (left part of the Figure 3) is more or less straightforward (e.g. *subClassOf* specifications are transformed to new child nodes within the tree), the aggregation view requires implementation of sophisticated programming logic. A new node in the aggregation tree is created when there is one (or more) object-property which has a domain-relation to the selected class AND has a range definition to another ontology class. The methods *buildAggregationTree(…)* and *analyzeNewRange(…)* of the class *OWLTreeConverter* realize this processing. Using the *prefuse* functionality called *Decorators* for tree edges the label of each object property can be displayed in the right part of the editor window (Figure 3).

### 2.4.6 Editing the SEMANCO ontology

New ontology elements, such as classes or object properties, can be created and added to the SEMANCO ontology by means of methods *addNewClass(…)* or *addNewProperty(…)* of the *MainModel* class. Next to these methods there are functions which enable modification *changeClassName(…)* and deleting *deleteProperty(…)* of ontology elements.

Editing of elements can be divided into three different steps. Firstly, the *Jena* ontology model is edited, for example with *OntModel.createClass(…)*. Secondly, the *prefuse* visualisation is updated, using *Tree.addChild(…)* for creating new classes or *Graph.removeNode()* for deleting properties. Finally, the added, deleted or edited elements have to be stored in OWL. This is accomplished by methods of the *SaveToOWL* class, were all changes to ontology first are registered by calling the *registerChange(…)* method and then saved to an OWL file using the *JDOM* framework. The *XMLWriter* class performs the actual file modification, at which there is no need to distinct between XML and OWL commands due to their similar document structure.

### 2.4.7 Visualisation of the ontology

Within the SEMANCO ontology editor the *MainGUI* class provides all necessary methods for displaying and modifying ontology. The editor's GUI optimises representation to the screen resolution of the target computer. Furthermore, the size of each view (subsumption –left part in Figure 3- and non-subsumption – right part of Figure 3) can be adjusted individually. An integrated side bar, which contains the control panel offering settings for tree orientation, tree depth and legend information (Figure 5), can be suppressed as well as customised.

In both views, a zoom function is available controlled by user's mouse-wheel. Using the methods *expandAndFocusClass(…)* and *autoPanAndZoom(…)* of the class *MainController* trees in both views are automatically expanded or collapsed by clicking a tree node; in addition the subsumption view comes with an animation for centering each concept that is in focus by the user.

Modification functions such as adding, deleting or changing of ontology elements can be accessed by a context pop-up menu on pressing the right mouse button. Due to their rather complex processes, these functions are implemented in a separated class called *PopupMenuController*.

A search option enables searching for a specific ontology class and provides auto complete functionalities. Its visualisation is realized in the class *SearchGUI*, auto complete is implemented in its inner class *AutoCompleteListener*.

# 3 SEMANTIC DATA EXPLORER

One of the main goals of the SEMANCO project is to integrate data from different sources in order to carry out energy analysis with the final goal of reducing $CO_2$ emissions in cities. The data sources have been modelled and integrated using semantic technologies according to the semantic energy model developed in the project. The Semantic Energy Information Framework (SEIF) provides federated access to the different data sources by means of a SPARQL endpoint. The main recipients of this data are the energy analysis tools developed in WP 5 Integrated Tools. These tools retrieve the data from the SEIF to analyse them or interface them to other energy analysis tools (for example, URSOS).

Since the data is exposed by the SEIF in RDF through a SPARQL endpoint, it is worth providing a user-friendly environment which enables users to explore the data and its relations without directly using the energy analysis tools. The main requirements for such an environment are:

- It should use the semantics of the data to navigate through data sources and their relations. It means that the user should use the SEMANCO ontology terminology and properties to define the queries to retrieve the data.
- It should provide a user friendly interface enabling non ontology skilled people to explore the data sources using natural language. According to the previous requirement, it is important to use the ontology to query the data. However the user should not see any ontology constructs. The annotation properties of the ontology are a valuable resource to be used by the semantic data explorer environment, since they describe the ontology components in a natural language.
- It should access data using SPARQL language through and the SEIF endpoint.

The research community has developed some tools to explore data which might fulfil the requirement stated above. The following list describes some of these tools:

- **Pubby** is a Linked Data Frontend for SPARQL Endpoints to display and enter data using a turtle look like interface. It is developed for Freie Universität Berlin and further details can be found in Cyganiak & Bizier (2008).
- **Relfinder** is a tool that detects and extracts relationships between objects. The result is showed in user interfaces that support the study and examination of the found relationships (Heim et al., 2010).
- **Cubee** is a knowledge-driven query formulation system for SPARQL generation. SPARQL queries are generated using an intuitive step-by-step process and are presented to the user in several visualisation interfaces (Asiee et al., 2012)
- **Nitelight** is a graphical tool for semantic query design which uses a Visual Query Language called vSparql. This tool enables users to create SPARQL queries using a set of graphical notations and GUI-based editing actions (Russell & Smart, 2008).
- **Graphite** is a system that enables the creation of a large query in a visual manner through direct manipulation, finding accurate and close matches, and visualising those (Chau et al., 2008).

The characteristics of these tools are summarised in Table 1. “Ontology engineers as final users” means that the user of the tool requires some semantic web knowledge to use it. ”Domains expert as final users” means that users without technical knowledge can use it. “Search for a specific concept” indicates whether a explicit concept, instance or literal can be

queried. “Data output personalisation” tells if data visualisation can be configured by the user. “Domain independent” specifies if the tool is not attached to a specific domain. “Find relations between instances” means if for two given instances, can be found the paths connecting both instances. And “Query formulations system” tells if the tool provides an ontology-guided method to write the queries.

*Table 1. Summary of the existing tools comparison*

| | Pubby | Relfinder | Cuebee | Nitelight | Graphite |
|---|---|---|---|---|---|
| **Ontology engineers as final user** | X | | X | X | X |
| **Domain experts as final user** | X | X | | | |
| **Search for a specific concept** | | | | | |
| **Data output personalisation** | X | X | | | |
| **Domain independent** | X | X | X | X | X |
| **Find relations between instances** | | X | X | | |
| **Query formulation system** | | | X | X | X |

As can be seen in Table 1, there is no tool which would cover all the required functionalities. Therefore, we found it necessary to develop a semantic data explorer environment which would meet all these requirements and implement the functionalities stated in Table 1. The main goal of this environment is to enable domain experts to use the ontology terms when navigating through data without having ontology engineering skills. It is also important to provide a querying system to enable users to make complex query sentences in order to find specific data in a guided way based on the data semantics. Finally, the environment should provide a personalisable user interface to be adjusted to different situations and domains.

## 3.1 The semantic data explorer components

The semantic data explorer environment has been designed as an interface –based on Semantic Web technologies– to enable domain experts and ontology engineers to explore the data accessible over the SEIF. It provides three different tools for each of the following forms of data exploration (Figure 6). The ***search*** tool retrieves the details of a specific data instance. The ***relate*** tool enables users to find paths between two data instances. Finally, the ***query*** tool facilitates users to define queries using the semantics behind of the data. The presentation of results is focused to help domain experts to interpret the data inside the ontology and using that to learn additional information (Figure 7). Though this environment has been developed and tested with the SEMANCO ontology with the use of data strongly related to the project, it has been designed to be customised for any other ontologies and data sets which follow Linked Data principles.

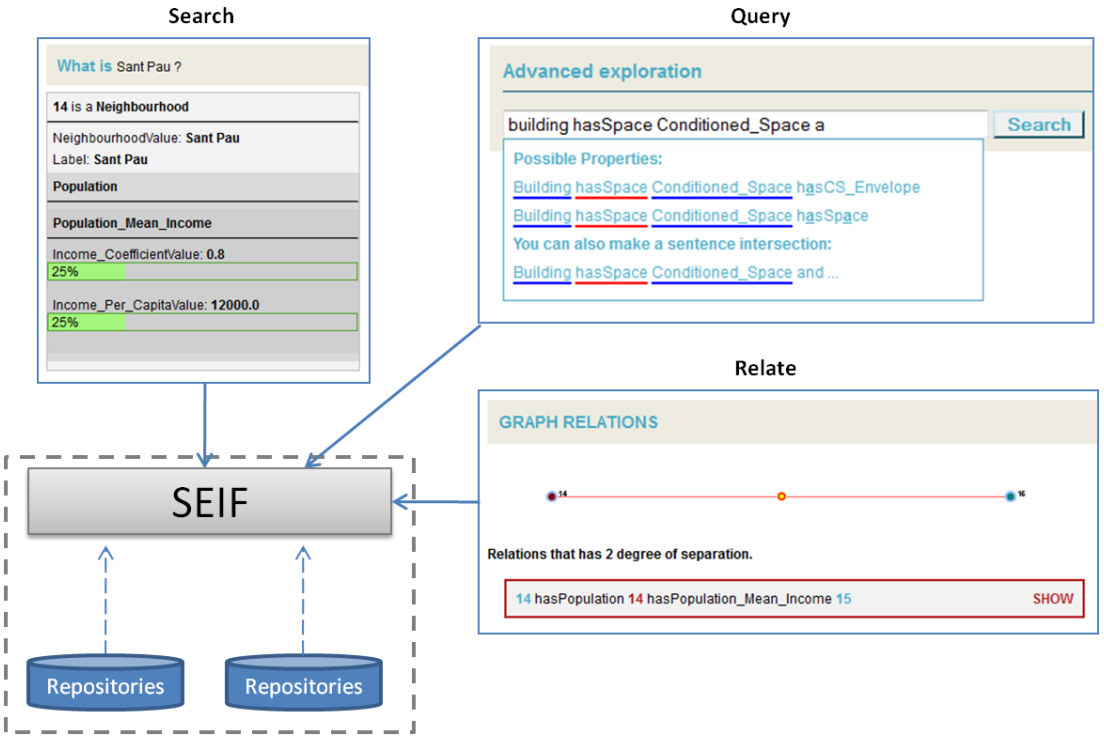


*Figure 6. Data explorer architecture*

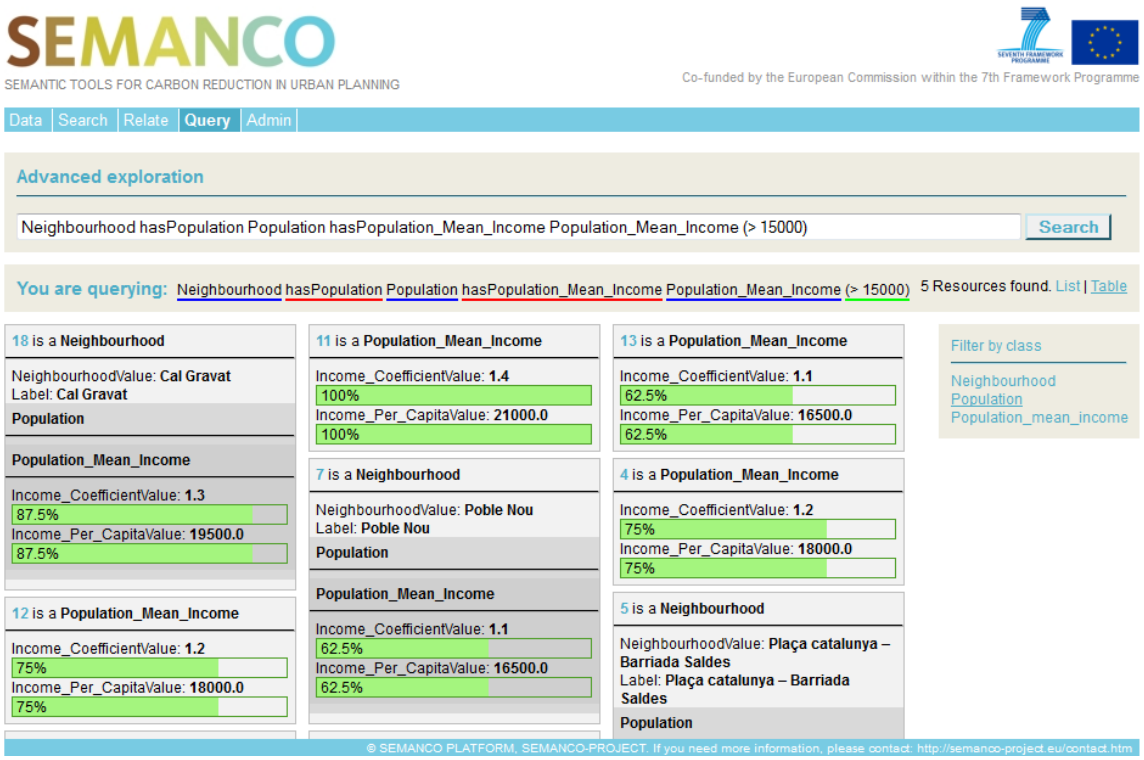


*Figure 7. Data Explorer interface*

The user-friendliness of the interface has been a key objective in the design of the environment. In the course of this, the annotations/metadata of the ontology elements play an important role in the presentation of data to the user, as well as in the interpretation if the users' queries.

### 3.1.1 The *search* tool

The ***search*** tool is used to retrieve specific instances with meet requirements formulated in the user input. The user introduces keywords which are sought in the properties of instances. A suggest feature has been implemented as a drop-down list which is activated while the user is typing the input. The drop-down list is filled with the matches and their corresponding types. The user can select one of the matches to retrieve data. Data values that match the keywords are returned to the user as a list. The instances can be selected to view its properties and relations. Once one match is selected, a list of its data properties and object properties is shown. Also, a graph is displayed which contains all instances related to the selected one. The graph can be used to navigate through the relations.

### 3.1.2 The *relate* tool

The ***relate*** tool finds relations between two instances and returns a list of paths that connect them. The inputs are two instances which are selected by the user in much the same way as with the ***search*** tool. The ***relate*** tool returns two kinds of information:

- a list with all the instances that match with the input words in a similar way to the ***search*** tool. These instances can be selected to explore their data.
- a graph including the paths between the two selected instances. The graph is composed of the nodes which represent the instances and edges which are the relations between instances. The nodes can be selected to highlight their related paths.

### 3.1.3 The *query* tool

The ***query*** tool supports a syntactic search method whereby the user is guided trough the ontology structure. The input of the tool is a phrase composed of two elements: concepts which are ontology classes (in terms of OWL) and relations which are properties that connect the classes. The ***query*** tool returns a list of sequences of instances which follow the same structure of the input phase. In Figure 8 an example is shown. The input phrase is composed of three classes that are connected with two properties. Then, the output provided by the ***query*** tool is composed of two sequences of instances; each sequence has the same structure of three classes and two properties. The user can filter the output, for example by providing a specific instance as a filter criterion.

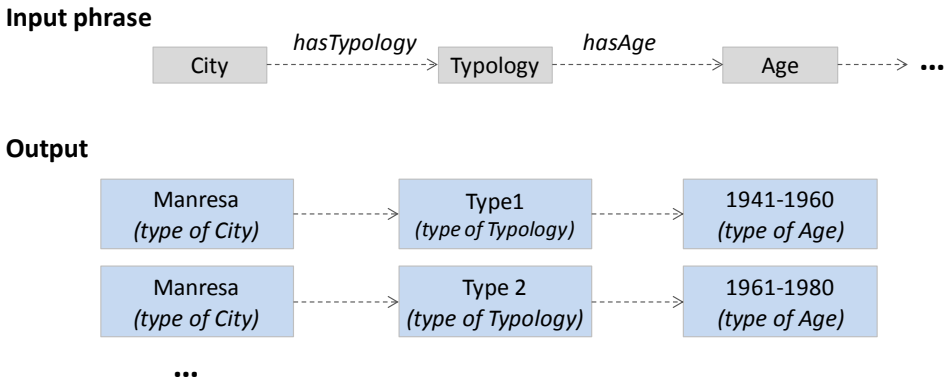


*Figure 8. Input and output example of the **query** tool*

To help the construction of the query, a suggest method has been implemented as a drop-down list showing if the current query is well-formed to be executed and also includes suggestions of other related phrases.

The results are displayed in two representations:

- a list which shows the instance sequences as human-readable sentences that can be read horizontally like sentences. The instances of the sequence can be clicked to show the related data. This option opens a pop-up window which contains the related data and a navigable graph with all its related instances.
- a table showing all instances contained in the different sequences breaking its structure. This representation is useful to see all the instances details at the same time. The instances can be filtered by their type.

### 3.1.4 Interface

The interface of the ***search*** tool is divided into two sections (Figure 9). The first section at the upper part of the window contains the input field where the user can enter terms that will be used in the search process. The second section below shows the data retrieved from the SEIF by the ***search*** tool. In the left part of this section, there is a list of the instances that match with the word written in the field (upper part). At the center, the detailed data related to the instance selected are displayed. Finally in the right side, a graph with the relations of the instance is shown. When a relation is clicked, the details about the relation are shown below the graph.

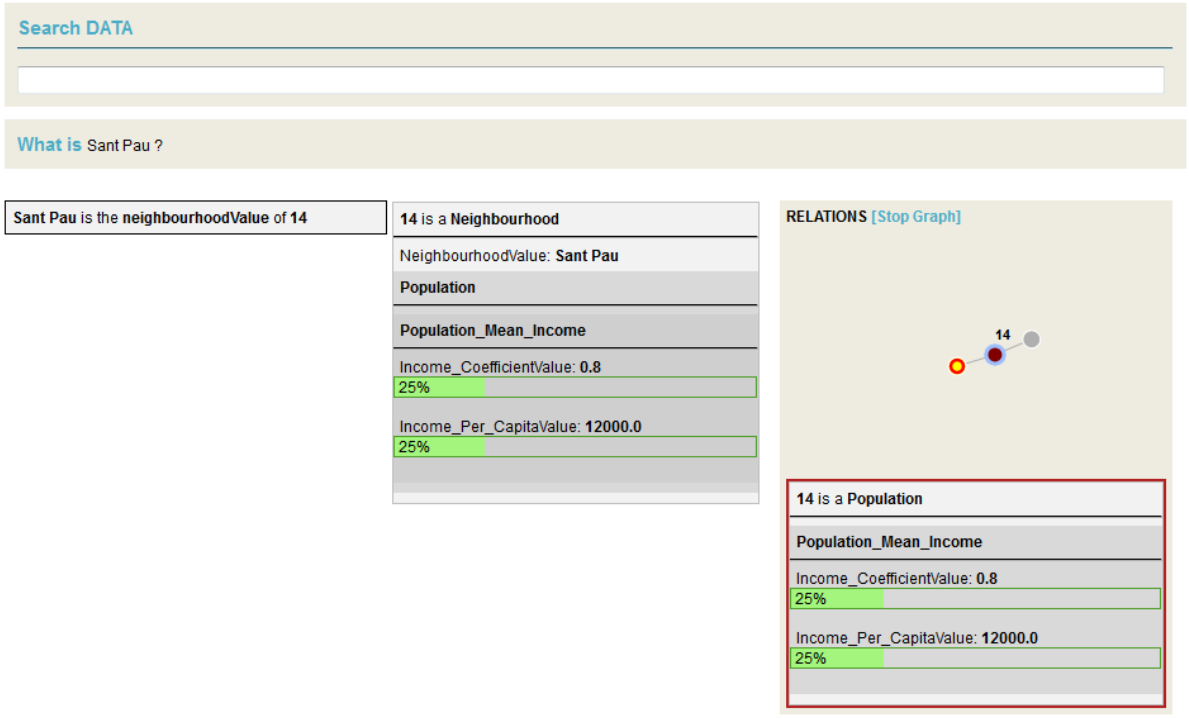


*Figure 9. The **search** tool interface*

Figure 10 presents the suggest method to guide the user. When the user starts to type a word, a drop-down list is displayed with the specification of instances that match the keyword entered in the field. The specifications include types of the instances.

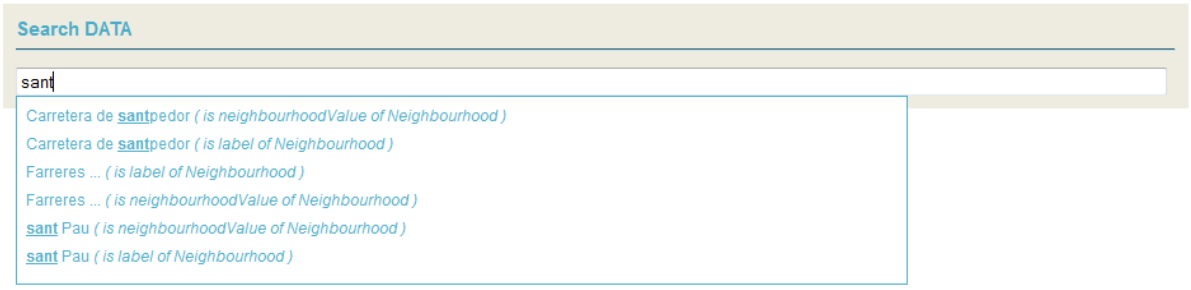


*Figure 10. Suggest drop-down list*

The interface of the ***relate*** tool is divided into three sections (Figure 11). The first one in the upper part of the window contains the input boxes where the user can enter two instances to search for relations between them. The second section shows information about the selected instances with their descriptions and a graph of relation to other instances. This works similarly to the ***search*** tool interface. The last section, at the bottom, shows a graph with the all possible paths that connects the two instances. If one of the paths is selected, its details describing the relation are displayed under the graph.

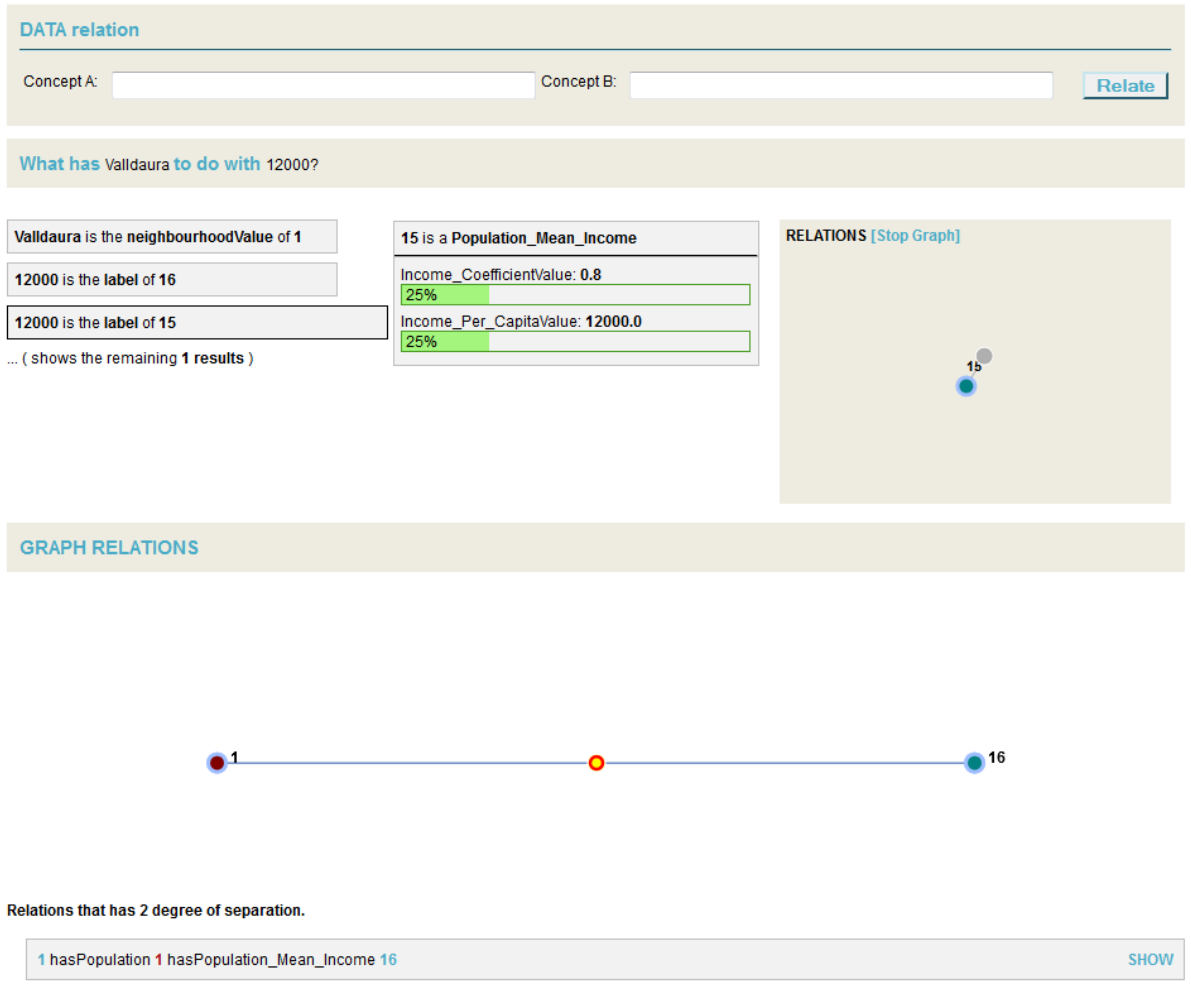


*Figure 11. The **relate** tool interface*

The interface of the ***query*** tool is divided into two sections (Figure 12). The first section in the upper part of the window contains the input box where the user writes the input phrase. While the user enters some characters, a drop-down list will display the elements that can be used to create the phrase. The next section shows the output retrieved from the SEIF. The output contains a list of sequences of instances that follow the pattern of the phrase. This information

can be represented in a list or in a table. In the list view, all the data of each instance can be shown under each sentence.

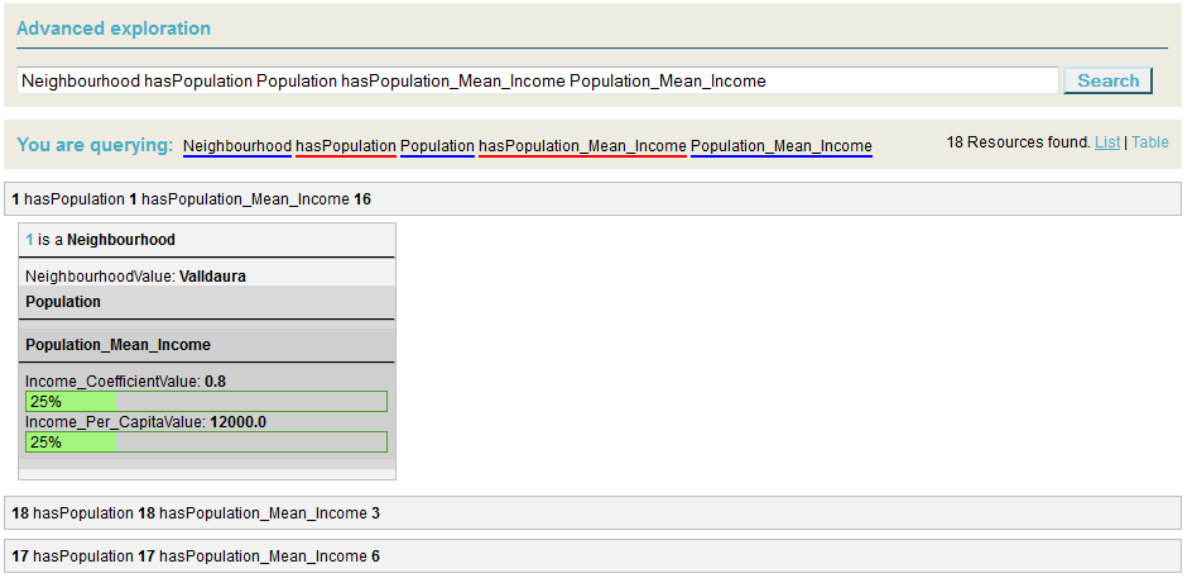


*Figure 12. The **query** tool interface*

A pop-up window that shows detailed information about an instance can also be opened by clicking on an instance name, in bold (Figure 13). The pop-up contains two sections, at the left all the information about the selected instance, and at the right a graph with its relations to other instances. The nodes of the graph can be clicked to show more details.

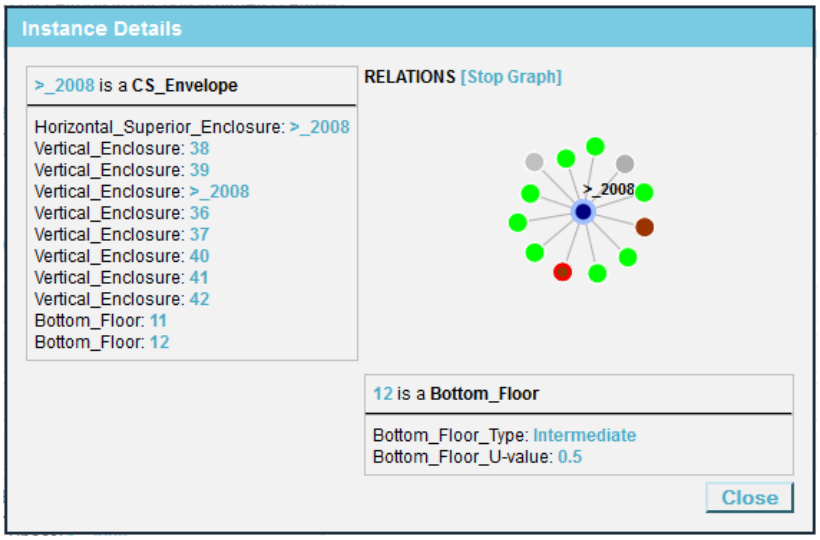


*Figure 13. Instance pop-up data*

### 3.1.5 Administration module

The administration module has been implemented to enable administrators to determine how the data is retrieved from SEIF and how it is shown to the user. This module enables experts to configure the SPARQL endpoint, to select the annotation properties required to make the interface more user-friendly and to choose query optimization settings. The module is divided in three settings sections: General Seetings, Class Settings and Color Setttings.

The General Settings section has the following fields:

- Endpoint: URI used to identify the SPARQL endpoint to make the queries.

- Graphs: URIs used to identify the Graphs to make the queries.
- Autocomplet Concepts: URI of the annotation property that provides human-readable names for concepts. Default value: rdfs:Label.
- Autocomplet Object Properties: URI of the annotation property that provides human-readable names for object properties. Default value: rdfs:Label.
- Autocomplet Data/Annotation Properties: URI of the annotation property that provides human-readable names for data and annotation properties. Default value: rdfs:Label.
- Autocomplet Instances: URI of the annotation property that provides human-readable names for instances. Default value: rdfs:Label.
- Class Uri: URI used to identify Classes. Usually it is set to rdf:Class or owl:Class.
- Object Property Uri: URI used to identify object properties. Usually it is set to rdf:Property or owl:ObjectProperty.
- Data Property Uri: URI used to identify data properties. Usually it is set to rdf:Property or owl: DatatypeProperty.
- Use rdf:domain and rdf:range to find relations: This option determines how the relations are obtained. True for using domain and range properties, or false, using the *DL-Lite*$_A$ axioms.
- Use implicit inverse relations: It takes into account non declared inverse relations in the ***query*** and the ***relate*** tool. It has a performance penalty.
- Use utf8_decode to present the results: It indicates if the tools are going to use the utf8_decode to solve the string matching process. Some triple stores, like virtuoso need this option.

In the Class Settings section the user can specify for each class how instances are going to be presented to the user. It is done by configuring each data property and object property (Figure 14). First, the user selects from the list box the class to be modified. Then for each property chooses the visualisation type. There are four options:

- Show: The property is shown in the table.
- Graph: The property which has numeric values shown in a graph bar. (Only for data properties)
- Hidden: The property is not shown inside the table.
- Open: The visualisation of the target class of the object properties is embedded inside the table. (Only for object properties)

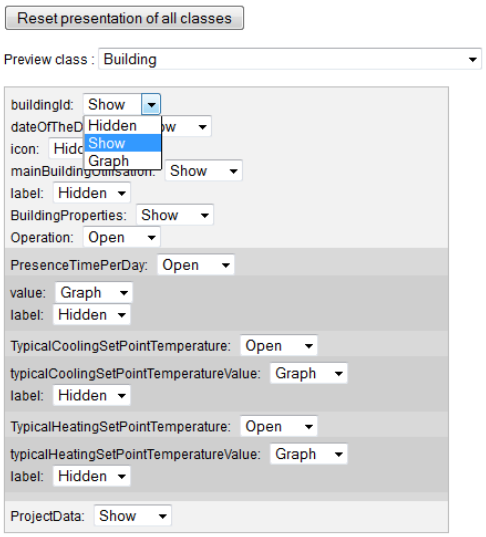


*Figure 14. Admin class presentation section*

In the Colour Settings section the user can define a colour for each class to be used in the graph representations (Figure 15). This colour is applied in the ***search*** tool graph and the ***relate*** tool graph to help the user to identify different kinds of instances.

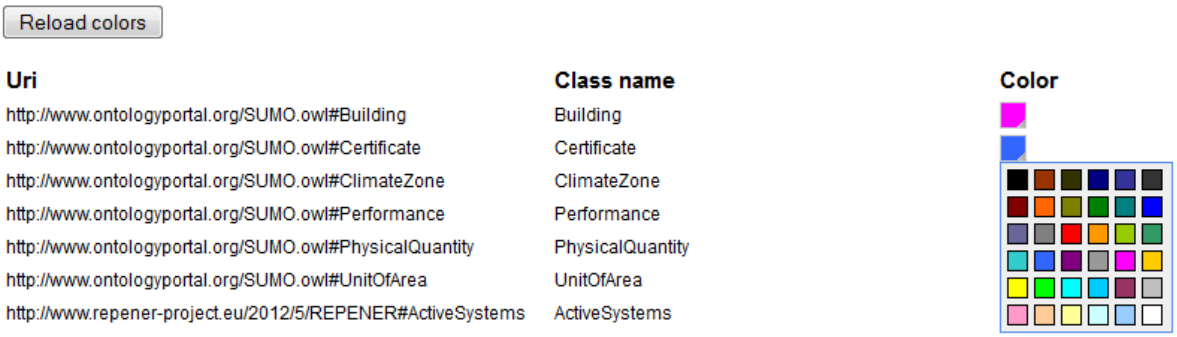


*Figure 15. Admin colour section*

## 3.2 Query engine

The query engine uses the input of the user to create a SPARQL code dynamically to retrieve data from a SPARQL endpoint. The generated code depends on the admin configuration. For example, the SPARQL query structure used to find relations between terms is shown in Figure 16.

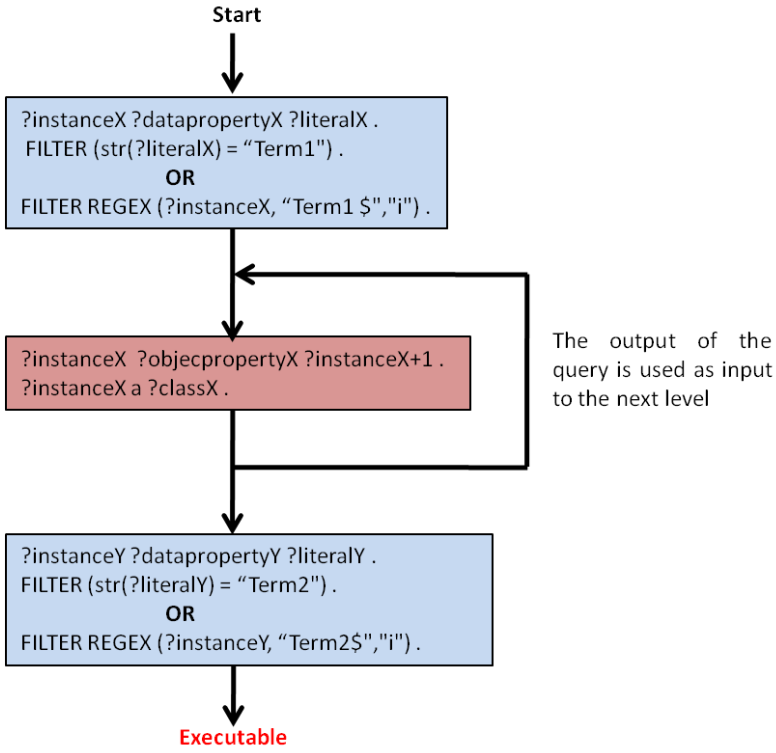


*Figure 16. Sparql code diagram to retrieve relation between two instances*

The blue blocks contain conditions that mark the beginning and end of the search code (Figure 16). Both blue blocks filter the instances by the keywords given by the user. The red box is used to find the paths between the instances by repeating n times, where n is the maximum length of the paths. To improve the performance, it has been designed that the length of the path is set from 1 to N while the system do not receive a timeout. This limits the size of the graph, but it improves the response time of the first relations levels.

Figure 17 shows the how the ***query*** tool works to execute a phrase provided by the user. The blue block is used to identify the classes that are in the phrase. The green block filters the instances that belong to the class and includes its data properties. The use of this block is optional. The red block is used to connect instances through the object properties. This block should be executed between two blue blocks since it works as a connector.

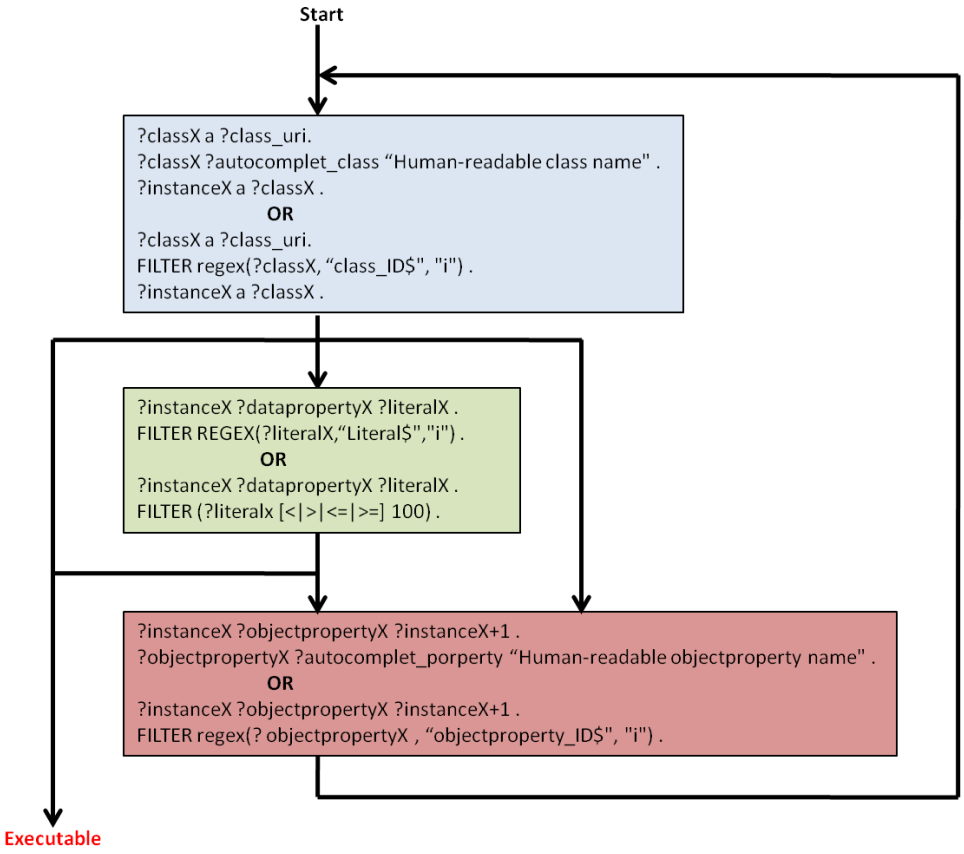


*Figure 17. Query Sparql code diagram*

## 3.3 Technical aspects of the implementation

The semantic data explorer has been developed as a web environment using HTML, JavaScript, CSS and PHP languages. The *Codeigniter* framework for PHP has been used to help the programming. The MySQL database is used to store the admin configuration and the cache data. Open source PHP libraries have been used to improve the functionality: ARC to parser RDF files, *Vivagraphjs* to implement the graph representations and *Jquery tiler/colorpicker* to improve the usability of the admin interface.

## 3.4 Examples of use

The semantic data explorer can be found at:

**http://arcdev.housing.salle.url.edu/semanco/data/**

This environment will be integrated in the SEMANCO platform which is being developed in the tasks of WP5.

A set of examples are described in the following sections which demonstrate how the environment works. The examples retrieve data provided by the partners in the different case studies.

### 3.4.1 Example 1

The user of this example wants to know what is "Carretera de Santpedor". To answer this question the most suitable way to solve it is through the ***search*** tool. The user types the name in the input field and the drop-down list is shown with some options. Once an element from the list is selected, its details are shown, and then the user can see that the "Carretera de

Santpedor" is a neighbourhood with some properties such as income per capita of 18000 Euros per year. The output of the tool is shown in Figure 18.

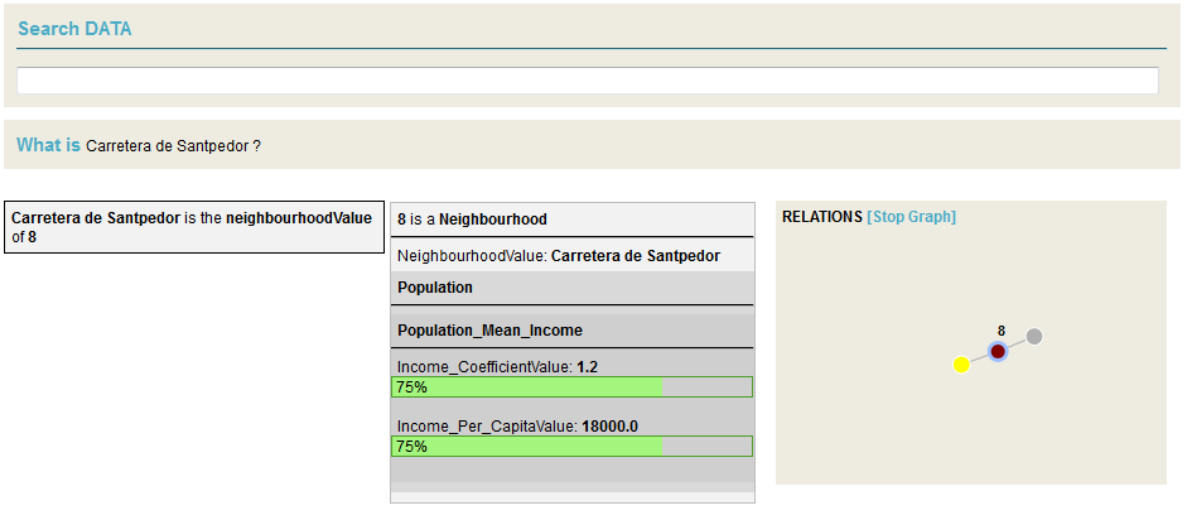


*Figure 18. The **search** tool example*

### 3.4.2 Example 2

In the second example, the user wants to find "buildings with a space heating system efficiency greater than 80%". In this case the ***query*** tool is the most suitable to find this kind of information. As long as the user is writing the phrase, it will be analysed by the tool providing some suggestions and alternatives. In the case of the example the phrase would be:

**Building hasSpace_Heating_System Space_Heating_System**
**hasSpace_Heating_System_Efficiency Space_Heating_System_Efficiency ( > 80 )**

If it is configured to use annotation properties instead of the class labels, then the phrase would be more naturally described like this:

**Building has Space_Heating_System has Space_Heating_System_Efficiency ( > 80 )**

As can be seen, the input phrase has the same structure as the ontology has. The outputs provided by the ***query*** tool are shown in the Figure 19.

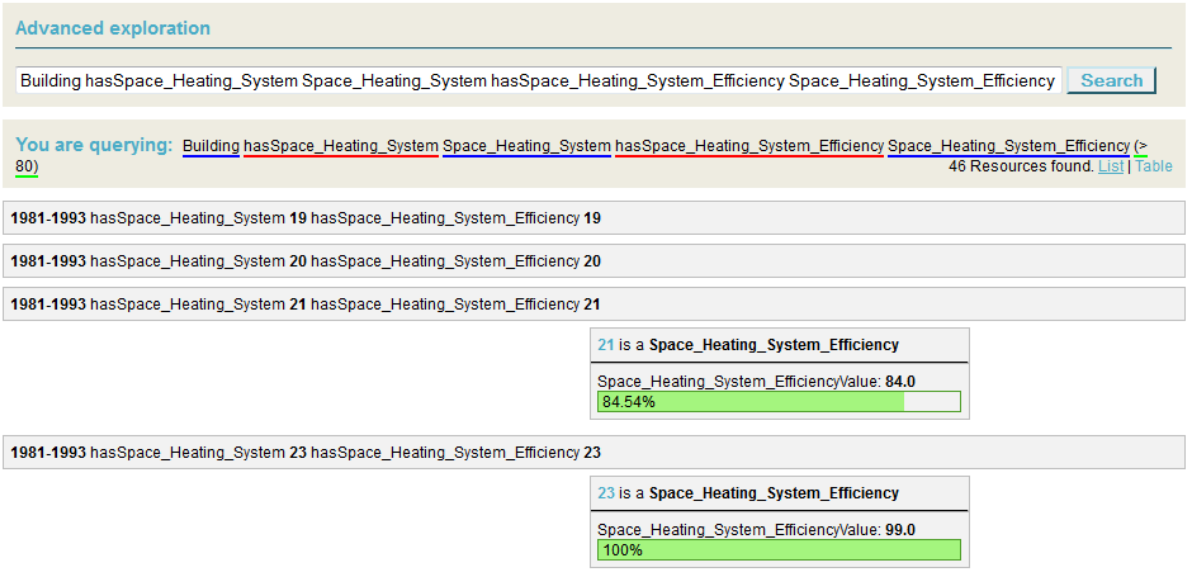


*Figure 19. Second example using the **query** tool*

### 3.4.3 Example 3

In the third example, the user wants to retrieve "buildings which have a u-value for walls greater than 2 and a percentage of windows greater than 10" (Figure 20). This is a more complex query than the second example since it is need to use the logical operator "and". This operator helps to find relations between two different phrases that have some related concept. In this case the required phrase would be:

**CS_Envelope hasBottom_Floor Bottom_Floor hasBottom_Floor_U-value Bottom_Floor_U-value (>2) and CS_Envelope hasVertical_Enclosure Vertical_Enclosure hasPercentage_Of_Window Percentage_Of_Window (>10)**

Would it be configured by using annotation properties instead of the class lables, the phrase would be more natural, for instance:

**CS_Envelope has Bottom_Floor has Bottom_Floor_U-value (>2) and CS_Envelope has Vertical_Enclosure has Percentage_Of_Window (>10)**

Figure 20 shows the query answer generated by the system after the query in its form above has been entered by the user.

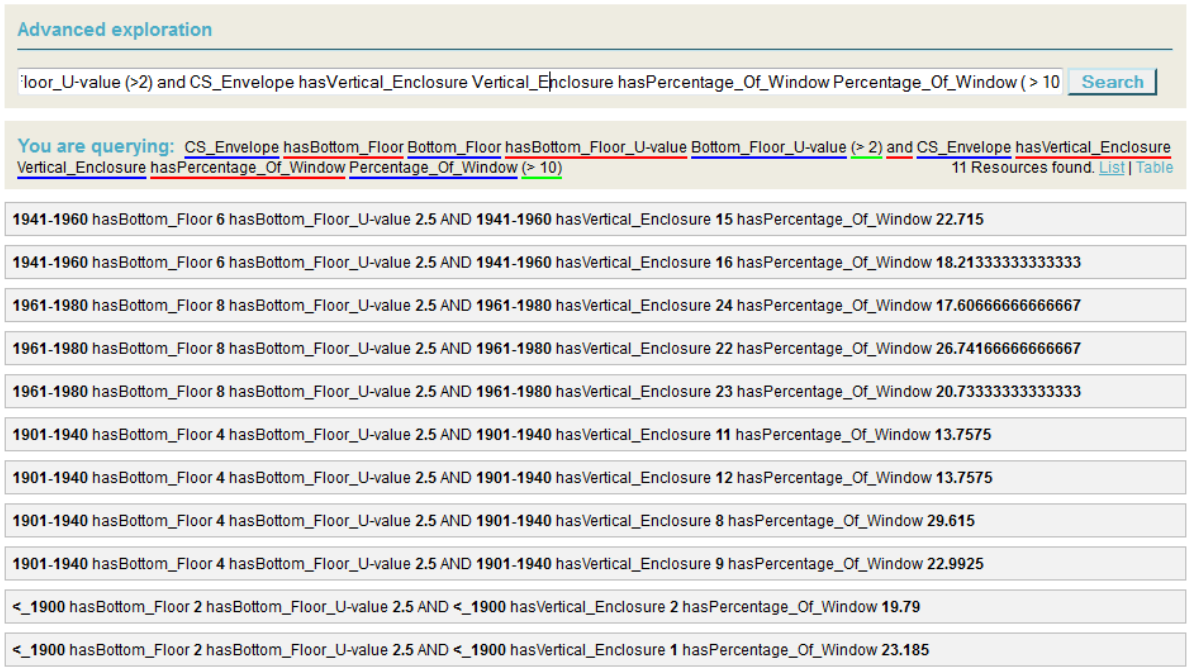


*Figure 20. Third example using the **query** tool*

# 4 CONCLUSIONS

## 4.1 Contribution to overall picture

The SEMANCO ontology editor has been designed and implemented to help users to code the ontology which is an important step of the ontology design methodology. The editor hides the complexity of *DL-Lite*$_A$ axioms enabling users without advanced ontology design skills to participate in the ontology building process.

The semantic data explorer is an important tool which gives visibility to the data exposed by the SEIF. It provides a generic way to explore the data using the semantic energy model created in the project. The user can query data distributed in different sources by entering keywords. Based on these keywords, formal queries are generated in automated manner. The answers to the users' queries are presented as a text and graphically or diagrammatically to make clear relations between concepts and instances contained in the answers.

## 4.2 Impact on other WPs and Tasks

The work done in this task is related to the previous Task 4.2 since the ontology editor has been used to code the semantic energy model. Moreover, the editor will be used for further elaboration of the energy model during the remaining lifetime of the project and beyond it.

The semantic data explorer enables domain experts to explore the data which is provided by the SEIF. It assists data providers in navigating through the data to be used in the case studies. It is already available online and it will be one of the components of the SEMANCO integrated platform being developed in WP5.

## 4.3 Contribution to demonstration

The ontology editor described in this document will help users to create the semantic energy model which is a fundamental part of the data integration process aimed to solve the interoperability issues between heterogeneous data sources.

Though the semantic data explorer it is not directly related to any use cases developed in the project, it will be integrated in the SEMANCO environment and help to the end user in the application of the analytic tools as a facility for exploration of the data provided by the SEIF.

## 4.4 Other conclusions and lessons learned

Both environments, the ontology editor and the semantic data explorer, have been designed and implemented to be domain independent:

- The editor can be used to design ontologies that conceptualise other domains than $CO_2$ emissions of buildings and cities. The only requirement is that the ontologies would be written in *DL-Lite*$_A$ formalism.
- Also the semantic data explorer can be used with other ontologies and data than the one used in this project, providing that both data and ontologies are exposed as SPARQL endpoints.

# 5 REFERENCES

Arpírez, J.C., Corcho, O., Fernández-López, M. & Gómez-Pérez, A. (2001) WebODE: a scalable workbench for ontological engineering. Proceedings of the *1st international conference on Knowledge capture*. 6-13, ACM New York, USA.

Asiee, A., Doshi, P., Minning, T., Sahoo, S., Parikh, P., Sheth, A., & Tarleton, R. (2012) From Questions to Effective Answers: On the Utility of Knowledge-Driven Querying Systems for Life Sciences Data. *Knoesis Center Technical Report*.

Chau, D., Faloutsos C., Tong, H., Hong, J., Gallagher, B., & Elliassi-Rad, T. (2008) GRAPHITE: A Visual Query System for Large Graphs. Proceedings of the *2008 IEEE International Conference on Data Mining Workshops*, pp. 963-966.

Cyganiak, R., & Bizer, C. (2008) Pubby - A Linked Data Frontend for SPARQL Endpoints. http://www4.wiwiss.fu-berlin.de/pubby/ (Accessed: 21/4/2013).

Heim, P., Hellmann, S., Lehmann, J., Lohmann, S., & Sregemann, T. (2010) Relfinder: Revealing Relationships in RDF Knowledge Bases. Proceedings of the *4th International Conference on Semantic and Digital Media Technologies (SAMT 2009)*, pp. 182-187. Springer, Berlin/Heidelberg.

Kapoor, B. & Sharma, S. (2010) A Comparative Study Ontology Building Tools for Semantic Web Applications. *International journal of Web & Semantic Technology (IJWesT)*, Vol.1, Num.3.

Khondoker, M.R. & Mueller, P.: Comparing Ontology Development Tools Based on an Online Survey. Proceedings of the *World Congress on Engineering 2010*, Vol. I, London, U.K.

Knublauch, H., Fergerson, R.W., Noy, N.F. & Musen, M.A. (2004) The Protégé OWL Plugin: An Open Development Environment for Semantic Web Applications. Proceedings of the *Third International Semantic Web Conference*, Lecture Notes in Computer Science, pp. 229-243, Hiroshima, Japan.

Nemirovski, G., Sicilia, A., Galán, F., Massetti, M. & Madrazo, L. (2012) Ontological Representation of Knowledge Related to Building Energy-efficiency. Proceedings of the *Sixth International Conference on Advacnes in Semantic Processing*, Barcelona.

Poggi, A., Lembo, D., Calvanese, D., De Giacomo, G., Lenzerini, M. & Rosati, R. (2008) Linking data to ontologies. *Journal on Data Semantics*, pp. 133–173.

Russell, A., & Smart, P. (2008) NITELIGHT: A Graphical Editor for SPARQL Queries. Proccedings of the *7th International Semantic Web Conference (ISWC 2008)*, Karlsruhe, Germany.

Sure, Y., Erdmann, M. Angele, J., Staab, S Studer, R. & Wenke, D. (2002) OntoEdit: Collaborative Ontology Development for the SemanticWeb. Proceedings of *First International Semantic Web Conference (ISWC 2002)*.