Comparative study of some free ontology editors

I. INTRODUCTION

The World Wide Web is in a continuous evolution. In an attempt to highlight developments in this area, certain stages in this evolution have been defined, namely Web 1.0, Web 2.0, Web 3.0, Web 4.0.

In the beginning, Web 1.0, the main purpose of those who create web pages was to provide information. The content was static, stored in files and folders on the server, there was no interactivity with the visitors. Styling was not separate from the text, tables and frames were used to place the content in the page. Browsers did not promote a tags standard, leading to incompatibility issues and data from in a form should have been mailed. The need for interaction will be the one that will make the transition to the next stage.

Web 2.0 or social web has been marked by user participation in the creation and distribution of text or multimedia content. Social networking platforms allowed users to share their thoughts and experiences by interacting with each other. Web pages have become dynamic, providing personalized content integrating the result of querying databases. Languages become standardized.[1]

Currently (Web 3.0) we talk about web semantics, a way for applications to understand the meaning of information on the web. Other features include the use of artificial intelligence with natural language processing, 3D Graphics with three-dimensional design u biquity with development of mobile devices and internet access everywhere [2].

Which will be the next stage in the evolution of www (Web 4.0)? Here the opinions differ. It will be Web of Things, which is a subset of the general concept of Internet of Things, that information flow will be highly personalized, augmented and virtual reality will have a far greater impact on users and social interactions or q symbiotic Web, considering that individuals and commercial enterprises are mutually dependent. [3]

Returning to the current state of the web, the focus has now been on enhancing metadata, data that adds significance to existing data. It is intended that WWW to be a provider of information and not just data. In this regard, it is necessary that the data gathered from different locations to be easily understood by the software agents.

To achieve this goal, a standardized representation of knowledge is required.

Ontology is a way of representing knowledge. There are many tools for creating, editing, and viewing ontologies, and it is a question of choosing the most appropriate tool for the ontology according to the complexity of ontology and user-owned knowledge.

The aim of this study is to provide a comparative view of the best known free tools for editing ontologies. Such comparative studies have been made in the past, but in view of the continuous evolution of the instruments, comparative studies are required periodically.

For this comparison, in order to evaluate each tool, I have created a demonstrating ontology that attempts to model artificial intelligence algorithms, and I imported a existing ontology. Demonstrative ontology is described in Chapter 3. Editing tools and evaluation criteria are listed in Chapter 4. The evaluation tools are extensively analyzed in Chapter 5.

II. CONTEXT OF SEMANTIC WEB

The concept of semantic web was introduced by Tim Berners-Lee in an article in Scientific American [4]. The technologies underlying this vision, according to the famous cake layer, are : URI, XML, RDF, RDFS, SKOS, SPARQL, OWL.

URI (Uniform Resource Identifier) is a unique address through which any web resource can be identified.

XML (Extensible Markup Language) is a markup language that defines a set of rules for encoding documents in a format that is both human-readable and machine-readable. [5]

RDF (Resource Description Framework) is a framework for representing information on the Web . The semantic web needed this model to describe any resource in any domain on the web. RDF uses this model to break down information into small pieces in a format known as a triple : subject-predicate-object. Knowledge is expressed as a list of statements, each statement takes the form of this triple [6 pg. 20,25]. The subject, the predicate and the object in this model must be uniquely identified by URI. The subject and the predicate represent resources, while the predicate represents the relationship between them. Multiple triplets can form a graph in which the subjects and objects are the nodes, and the predicate edges. Several common serialization formats are in use, including:Turtle,N-Triples,N-Quads,JSON-LD,N3, RDF/XML, (first standard format for serializing RDF),RDF/JSON. It is also used for building vocabularies.

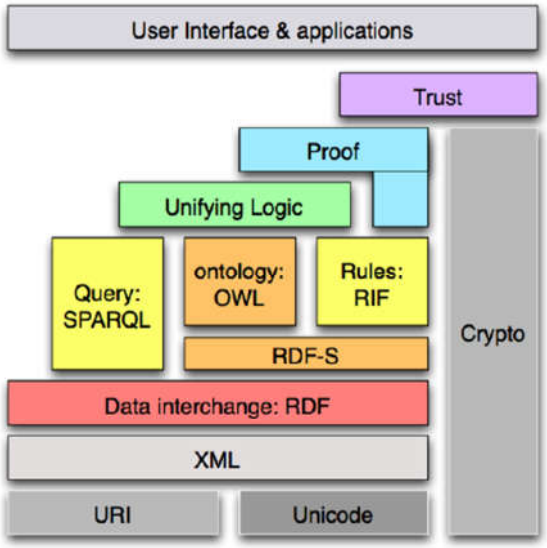

Figure 1 - A Revised Semantic Web Layer Cake

RDFS (Resource Description Framework Schema) can be seen as RDF's vocabulary description language, provides language constructs that can be used to describe classes, properties within a specific application domain [6 pg 111].

SKOS (Simple Knowledge Organization System) is a 'W3C recommendation designed for representation of thesauri, classification schemes, taxonomies, subject-heading systems, or any other type of structured controlled vocabulary'.[7]

SPARQL (SPARQL Protocol and RDF Query Language) is a semantic query language for databases able to retrieve and manipulate data stored in Resource Description Framework (RDF) format. [8]

OWL (Web Ontology Language) is a Semantic Web language designed to represent rich and complex knowledge about things, groups of things, and relations between things.[9]

An ontology is a formal description of knowledge in a certain area modeling entities, attributes, relationships and axioms. Ontologies are of several types, by the description level we have vocabularies (described by schema - databases, XML), taxonomies (described by RDFS), lexicon (SKOS modeling), relational system (database schema), axiomatic theory (axioms are used). By purpose and context of use we have specialized ontologies, general ontologies, intermediate ontologies[11 pg 27-36]

Ontologies can be expressed using different syntaxes:

- functional-style syntax - is generally used by OWL tools designers for specifications.
- RDF/XML - although it's difficult to read by people, is the only syntax that has to be supported by OWL tools.
- Manchester syntax - is a much easier to read and write representation, and aims to use it in editing ontologies in class expression from various OWL tools.[6, pg 160]
- Terse RDF Triple Language (Turtle) - is a series of triple semantics: subject - predicate - object, widely used by owl tools.
- Jason-LD - is based on Json format and tries to encompass in this Linked Data format.[12]
- OBO format - is a popular ontology file format. It can express a subset of the description logic language OWL-DL 2.0 but in addition has standard syntax for representing important classes of meta-data including as synonyms, references to publications and deprecated

IDs. It is designed to be human readable and editable, easy to parse, easy to extend and to have minimal redundancy [13]

- LaTeX is a document preparation system, based on TeX. Ontologies may be serialized in LaTeX for ease of reading, publication, etc.[14]

An ontology includes: categories, classes, fundamental concepts, properties associated with concepts, relationships between concepts [11 pg 46]. In OWL, axioms are basic statements of classes or individuals and properties.Classes, individuals or properties are also called entities. Expressions are combinations of entities which form complex descriptions about new entities. Expressions are a main reason why we claim OWL has a much more enhanced expressiveness compared to other ontology language such as RDFS [6 pg.159].

There are 2 versions of OWL: OWL1 and OWL2.

Class specification - classes designate categories of individuals. Classes are in relationships with the others - subordination, equivalence, disjunction, complementarity, reunion, intersection.

Properties specification - makes the connection between two classes (object property) or between a class and a data type (data property). Properties may have subproperties in the sense of specialization. properties can have some caracteristics:

- functional - the property has a single value for each individual.
- symmetric - the property linking two classes is valid in both directions
- transitive - if we have a chain of classes linked together with such property, the classes at the ends can also be linked to this property
- reflexive - describes the relationship of a class with itself
- inverse functional, asymmetric, ireflexive - are the inverse of the above.

Individual specification - individuals are of a certain type, that is, belong to a category (class).

Restrictions - we can use the properties to describe new classes. Restrictions can be quantifier restriction , cardinality restrictions and hasValue. Quantifier restriction ca be existential (some) si universal (only). Cardinality restrictions describe the number of values a property may have.

OWL2 contains new elements and implicitly new possibilities for expression. Some of them :

- expression of negation
- qualified cardinality restrictions
- the ability to express keys in the sense of databases
- specification of properties by composition[15][16]

III. ONTOLOGY FOR TESTING EDITORS

To estimate the power of each editor, besides the import of an existing ontology, we have also developed an ontology that models the search algorithms used by artificial intelligence.

Search plays a major role in solving many Artificial Intelligence (AI) problems. Search is a universal problem-solving mechanism in AI. In many problems, sequence of steps required to solve is not known in advance but must be determined by systematic trial-and-error exploration of alternatives.

The problems that are addressed by AI search algorithms fall into three general classes:

- single-agent path-finding problems,
- playing games,
- constraint-satisfaction problems

Single-agent path-finding problems

Often a problem can be formulated as a pathfinding problem in which the solution is represented by a series of actions that make the transition from the initial state to the goal state. Classic examples in the AI literature of path-finding problems are sliding-title puzzles, Rubik's Cube and theorem proving. Real-world problems include the traveling salesman problem, vehicle navigation.

Game playing

Adversarial search problems (a.k.a.games) are problems in which we have two or more agents and each one tries to maximize its outcome. Chess, Checkers, and Gomoku are such examples.

Constraint-satisfaction problems.

Are problems where the desired state has to meet a number of restrictions. Real-world examples of constraint satisfaction problems are planning and scheduling applications.

In order for a problem to be approached by an algorithm it is necessary to find a representation of the problem. Problem space is a set of states and the connections between them to represent the problem. Problem space graph is used to represent a problem space. In the graph, the states are represented by

nodes of the graph, and the operators by edges between nodes. Although most problem spaces correspond to graphs with more than one path between a pair of nodes, for simplicity they are often represented as trees, where the initial state is the root of the tree.

The solution to a problem can take the form of a set of values, an action or a sequence of actions, depending on what you want: optimization, best action or sequence of actions.

The environment may have several characteristics:

- it is totally, partially observable or unobservable - the degree of knowledge of the current state;
- deterministic or stochastic - the following state can be determined depending on the current state and an action or not;
- episodic or sequential - choosing an action influences the choice of other decisions;
- discrete or continuous - space states can be perceived as discrete units or continuous space; [19 pg 42-44]

Search algorithms can be categorized according to the type of search used:

- exhaustive search
- heuristic search
- iterative improvement (local) search
- adversarial search

Exhaustive search (uninformed or blind) - each node of the graph is analyzed until the goal node is found and the path is retained. The order in which nodes are checked differs depending on the algorithm.

Breadth-first search (BFS) is an algorithm for traversing or searching tree or graph data structures. It starts from the root node, explores the neighboring nodes first and moves towards the next level neighbors. It generates one tree at a time until the solution is found. It can be implemented using FIFO queue data structure. This method provides shortest path to the solution.

Depth-first search (DFS) is an algorithm for traversing or searching tree or graph data structures. One starts at the root (selecting some arbitrary node as the

root in the case of a graph) and explores as far as possible along each branch before backtracking.

Bidirectional Search searches forward from initial state and backward from goal state till both meet to identify a common state. The path from initial state is concatenated with the inverse path from the goal state. Each search is done only up to half of the total path.

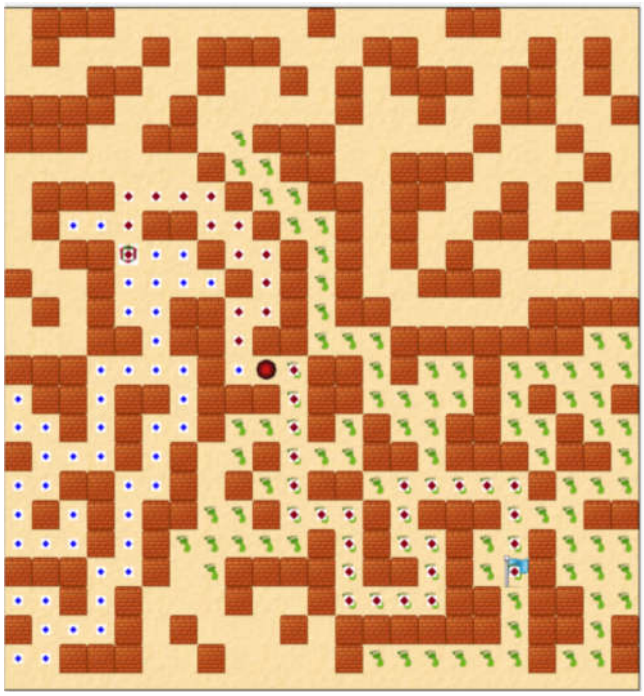

Figure 2 - Bidirectional Search use in a random maze – (simulation in Greenfoot)

Uniform Cost Search - sorting is done in increasing cost of the path to a node. It always expands the least cost node. It is identical to Breadth First search if each transition has the same cost. It explores paths in the increasing order of cost.

Heuristic search (informed) - try to reduce the number of nodes visited using the knowledge available to select the most promising nodes. for this, a heuristic function is used.

Greedy best-first search - expands the node that is estimated to be closest to goal, using Euclidean distance or Manhattan distance. As a disadvantage, it can get stuck in loops and is not optimal.

A* search is a combination of lowest-cost-first and best-first searches that considers both path cost and heuristic information in its selection of which path to expand. For each path on the frontier, A* uses an estimate of the total path cost from a start node to a goal node constrained to start along that path.

Iterative-deepening A∗ (IDA*) combines the benefits of BFD, DFS, and heuristics.

Simplified Memory Bounded A* (SMA*) - it prunes nodes whose expansion has revealed less promising than expected. [17] It expands the newest best leaf and deletes the oldest worst leaf [19 pg 101].

Iterative improvement (local) search - the most appropriate where the the pathj or the path cost is irrelevant, all that matter is the solution. it is also well suited for optimization problems where even if the optimal solution is not found, try to find one as close as possible to the optimum.

Hill-climbing search - select between the nodes at the border the one that has the optimal cost function - uphill. It works and if the environment has continuous space search. It can lock into a local minimum on a plateau or on the ridge. There is variants : Simple Hill climbing (choose the first node with a better cost), Steepest-Ascent Hill climbing (compare the cost of all nodes and choose the optimal one), Stochastic hill climbing (randomly chooses a node and decides whether or not to choose another) [20]

Simulated annealing - combine the hill climbing with a random choice. The choice of the next node is random and if it has a better cost it is chosen, otherwise it is only chosen with a certain prophecy, probability that can increase or decrease depending on the cost. [19 pg 125]

Local beam search - is an optimization of best-first search. Unfruitful searches are abandoned. To avoid concentration, there is also a stochastic version.

Tabu search - involves the existence of a memo with the nodes visited to avoid the paths already visited.

Genetic algorithms - combine an uphill tendency with random exploration and exchange of information among parallel search threads. Genetic algorithms resemble with local beam with a few differences : the following nodes are generated from the combination of two parent nodes and suffer small mutations, each generation is started with k nodes chosen at random and based on a fitness function.[19 pg 127,128]

Ant colony optimization (ACO) - is based on the ability of ants to find the shortest way (between nest and food). The ants population indirectly communicate their knowledge through the pheromones they leave behind. The intensity of pheromones is an indication of the choice of that path

Adversarial search - the purpose of the algorithms in this category is to find a strategy - which is the best option considering the possible moves of the opponents.

Minimax Algorithm - calculates based on a score the best move. This calculation can be made depending on the possible moves of the opponent "thinking" ahead with a few moves.

Alpha–beta pruning - is a search algorithm that seeks to decrease the number of nodes that are evaluated by the minimax algorithm in its search tree. It stops completely evaluating a move when at least one possibility has been found that proves the move to be worse than a previously examined move.

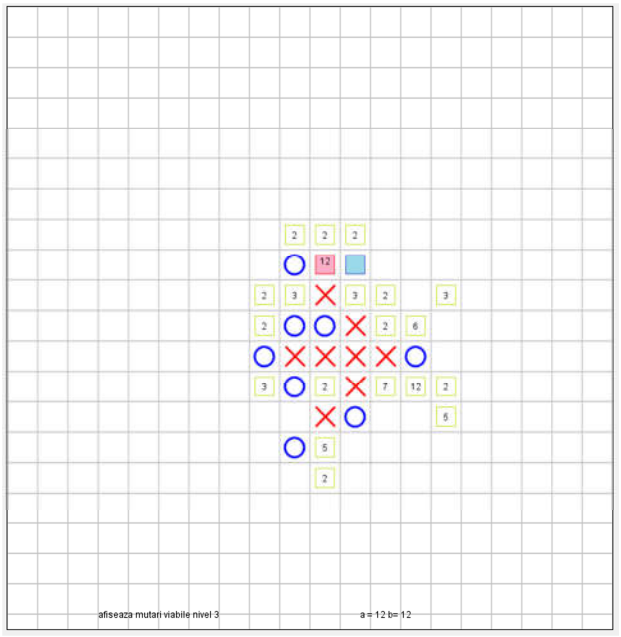

Figure 3 - Using Alpha-beta algorithm in Gomoku (simulation in Greenfoot)

Ontology classes :

- Problem (SingleAgentPathfindingProblem, GamePlaying, ConstraintSatisfactionProblem),
- RepresentationOfProblem
- Environment,
- Solution (Action, SequenceOfActions, SetOfValues)
- Algorithm
  - ❖ AIAlgorithm
    - ➢ SearchAlgorithm
      - ○ ExhaustiveSearch (BreadthFirstSearch, DepthFirstSearch,BidirectionalSearch,Uniform Cost Search)
      - ○ HeuristicSearch (GreedyBestFirstSearch, A*Search)

- IterativeImprovementSearch (HillClimbingSearch, SimulatedAnnealing, LocalBeamSearch, GeneticAlgorithms, AntColonyOptimization)
- AdversarialSearch (MinimaxAlgorithm, AlphaBetaPruning).

Object Properties :

- Problem - hasSubProblem (reflexive);
- Problem - isResolvedBy - Algorithm;
- Algorithm - resolves - Problem; isResolvedBy isInverseOf resolves;

Data Properties:

- Environment - (isObservable; isDeterministic; isEpisodic; isDiscret)
- Algorithm (isComplete, isOptimal)

Individuals:

- Problem (Maze, Gomoku),
- A*Search(IterativeDeepeningA∗, SimplifiedMemoryBoundedA*, StochasticHillClimbing)

IV. COMPARATIVE STUDY OF ONTOLOGY EDITORS

Tools considered

Protege is a free, java based, open-source ontology editor and framework for building intelligent systems. It is developed and maintained by Stanford University and it is supported by a strong community of academic, government, and corporate users.[21]

It is a tool which facilitates the creation, editing, viewing and querying of ontologies. Protege can import and export ontologies in various formats. The interface can be customized to create a comfortable work environment for users, support users in constructing and storing frame-based domain ontologies, customizing data entry forms, and entering instance data. It can be extended through a large number of plugins that improve various aspects of the application or bring new features (reasoners, visualizations, SWRLTab etc.).

WebProtégé is a free, open source collaborative ontology development environment for the Web which facilitates the creation, editing, viewing and sharing of ontologies. The interface which is simple and customizable, provides access to commonly used OWL constructs. It provides a history of revision and changes. It has support for editing OBO ontologies, alow upload and download for multiple formats of ontologies (supported formats: RDF/XML, Turtle, OWL/XML, OBO, and others)[21]

OWLGrEd is an ontology editor which allows creation, modifying and graphical visualization ontologies using a compact yet intuitive presentation. Its grahical visualisation is inspired by UML class diagram notation, but it also alows Manchester syntax for class expressions [22]

FluentEditor, according to their own description, is a comprehensive tool for editing and manipulating complex ontologies that uses Controlled Natural Language. Fluent Editor provides one with a more suitable for human users alternative to XML-based OWL editors. It's main feature is the usage of Controlled

English as a knowledge modeling language . Is automatically translated into and from description logic, OWL 2, SWRLSupported. [23][24]

Criteria considered

I took into consideration several evaluation criteria for ontology editors to provide a representative image of free tools for all user categories

- **hardware / software requirements** - an application may have higher hardware / software requirements but to offer more facilities with the risk to reduce the number of potential users while, a less demanding application is much more accessible but to offers fewer features. We considered on which operating systems can be installed, installation difficulty, disk space and memory requirements during running, other components that were previously needed on the computer.
- **interoperability** represents the ability of the application to use and export data from and to other applications. Is about the range of ontology serialization formats for import / export and export of graphical visualization.
- **support for graphical visualization** - graphical visualization for ontology is important to the human user (a picture makes a thousand words) especially when the authors want to present users a more representative image of ontology
- **support for reasoning** - the main functions of a reasoner are to verify the consistency of ontology and to infer new axioms based on existing ones.
- **suport for queries** – as in the case of databases, the possibility of extracting desired data from ontology highlights their usefulness.
- **extensibility** - the ability to add new functionalities allows the tool to be adapted to the needs of each user.
- **usability** - the ease with which users become familiar with the application interface, how intuitive it is, how easy find the desired item and how customizably is, in generally has effects on the quality of ontology and

time of completion. It also counts the options and suggestions offered during use, portability, the ability of other users to participate.

Editors analysis

Protege

- hardware/software resources
  - it can be installed on Windows, Mac OS X, Linux
  - is available as a downloadable archive that can also include JRE (Java Runtime Environment); in the case of version 5.5.0
  - for Windows, the archive has a size of 117 MB, reaching after unpacking to 218 MB. No installation required.
  - the memory used is limited to 444 MB.
- interoperability
  - offers the ability to import ontologies in various formats from disk or web or by specifying the URL.
  - the export of ontology can be done in several formats: RDF / XML, Turtle, OWL / XML, OWL Functional, Manchester OWL, OBO, LaTex, Jason-LD.
  - it also offers support for conversion between different formats. For example, you can import a file in RDF / XML format and then export it to Turtle format.
- support for graphical visualization
  - OntoGraf offers visualizations in several forms with the possibility of exporting as a picture or as a DOT.
  - has a number of plug-in for various other views that can be saved as an image. For example, VOWL
- support for reasoning
  - has a preinstalled reasoner (HermiT) but other reasoners can also be used (plug-in for Pellet, Fact++, RacerPro). HermiT uses direct semantics and passes all OWL 2 conformance tests for direct semantics reasoners[3].

- verifies the consistency of ontology and extracts by inference other relationships between classes.
- the reasoner can start or stop. If it is turned on, it will check if the ontology is consistent.

➢ support for queries

- there are three possibilities for querying ontology: DL query, existential query, SPARQL query.
- query possibilities can be extended - by installing a plug-in, you can use SQWRL for quering which is an extension of OWL / SWRL.
- other facilities
- it can generate documentation in HTML format.
- it can generate java code with classes and properties from ontology
- many plug-ins are available, some can be installed directly from the application, others can be downloaded and copied to the plug-in directory of the application

➢ usability

☐ creating an ontology

- when creating an ontology, it first establishes the IRI address of ontology with the ability to specify the version of ontology.
- In terms of adding prefixes, the application helps us by providing a list with the most used prefixes (owl, rdf, rdfs, xml, xsd) with the possibility of deleting or adding new prefixes.
- in the same view, annotations can also be added. The application helps us by providing a list of options (owl:versionInfo, rdfs: label, rdfs:comment, rdfs:isDefinedBy, rdfs:seeAlso) .

☐ creating classes

- surprises by intuitive way of creating / adding classes and subclasses.
- select a class and use one of the options - add a class on the same level (sibling), a subclass, or delete the class.
- you can duplicate a class and modify it.

- you can directly create a hierarchy of classes using indentation, one class per row, so you can build a classification without knowledge in ontology.
- in the annotation view can be made a description of the class. The application provides several owl and rdfs items for this purpose.
- OWLAx is a Protege plugin which allows the creation of classes, individuals and relationships between them in a graphic style similar to UML diagrams. for the classes and relationships thus created, the corresponding axioms can be generated and integrated into the current ontology of your choise. Graphics can be formatted in various ways - background color setting (including gradient), text alignment, graphic space, label and box rotation, shadow effect.

☐ adding object properties

- you can choose the domain and range with access to the class hierarchy.
- you can check the characteristics of the property: functional, transitive, symmetric, etc.
- you can specify certain properties - Equivalent With, SubProperty Of, Disjoint With, Inverse Of

☐ adding data properties

- resembles object properties - you can specify scope and range, with the difference that at range you can choose the data type from a predefined list or you can enter expressions.

☐ adding individuals

- you can choose the type in the hierarchy of classes, you can add object property assertion and date property assertion.

☐ importing an ontology

- importing an existing ontology is very simple from the File - Open menu. You can immediately see the prefixes used, the annotations, the classes, the properties and the individuals.

☐ reasoning

- to make inferencies and check the consistency of the ontology, the reasoner must be started. For infered relationships or

inconsistency of ontology, the reasoner provides the possibility to visualize the deductions used.

- to take into account changes made to ontology, there is a synchronization option. You can also configure the types of inferences displayed.
- I also tested the possibility to use another reasoner - for example I installed the Pellet plugin and selected it from the menu as a reasoner

☐ querying

- querying requires knowledge of the Manchester syntax or the SPARQL language.
- DL query uses Manchester Syntax to query ontology with the possibility to add results to ontology. You can query superclasses, equivalent classes, subclasses, and instances.
- SPARQL query - you find here the most common prefixes and the SELECT skeleton with a WHERE clause, which facilitates the query.

☐ visualization

- in Protege default visualization tool is OntoGraf that provides a image of classes and their relationships;
- in OntoGraf, the classes of interest are added by doubleclick on the class of the class hierarchy. With various mouse actions, you can see details about a particular class, the relationship between two classes, you can add subclasses and superclasses to a selected class.
- you can choose the layout of the graphic elements in space (radial, spring, tree horizontal, tree vertical etc.). The graph can be saved as an image, including as a DOT (graph description language).
- to compensate for the lack of a graphical view, there is a VOWL plugin that produces pretty good graphics. there are only a few possibilities to set the distance between the graphics and save the image only in the clipboard.

- others
  - Protege also offers a number of metrics - the number of axioms, classes, subclasses, properties, subproperties.
  - export option OWLDoc creates and exports documentation as HTML files that can be viewed in a browser.
  - there is an option to generates Java code for each class
  - refactoring - you can rename a class, move a class with all subclasses by drag and drop, rename an IRI.

Webprotege

- hardware/software resources
  - only a web browser, email account, and internet access is required
- suport pentru import/export
  - import ontology - you can import ontologies by providing a URL or a local zip archive
  - sharing - the project can be made public or shared with others who can view, comment or edit.
- support for graphical visualization
  - none
- support for reasoning
  - none
- support for queries
- other facilities
  - history with changes made
  - usability
    - the interface is organized on tabs, the most important being classes, properties, individuals, query. At most boxes, the application comes with suggestions as you start writing, making it easy to work and avoid possible writing errors. Also, the application is intended to follow the flows used by users to create various elements of ontology - Depending on the style of work, classes, properties, and individuals can be created in different ways.

- ☐ creating an ontology
  - you can create an ontology from scratch or from existing sources.
- ☐ creating classes
  - it is possible to fill in the name of the class, annotation (you can choose from a most frequently used preset list)
  - we can override superclasele (parents) or properties (relationship). If the properties do not exist, it creates them. Similarly, if the class to which the property refers does not exist, it creates it.
  - classes can be added, moved or deleted
- ☐ adding object properties
  - you can pass the property name, annotations, domain and range
  - properties can be added, moved or deleted
- ☐ adding data properties
  - you can pass the property name, annotations, domain and range
  - you can check whether the property is functional or not
  - properties can be added, moved or deleted
  - if the class that it refers to (domain) or the type of range does not exist, the application with the user's consent creates them.
- ☐ adding individuals
  - to determine the type of an individual are given several variants: if we created a class and we want to add individuals we use the Sync Selection button; if we add individuals later, we can add the type from the hierarchy of classes. There are also variants: jump to parent, jump to child, jump to sibling or we can search for class.
- ☐ importing an ontology
  - to import an ontology stored on disk we must provide a zip archive with a document called root-ontology.owl

➢ others

- sharing - you can set up a list of users who can access ontology and what actions they can do: view, comment or edit.

OWLGRed

- hardware/software resources
  - is available as a downloadable archive (.zip); in the case of version 1.6.9.3.
  - for Windows, the archive has a size of 67.4 MB, reaching after unpacking to 167 MB. No installation required.
- suport pentru import/export
  - offers the ability to import ontologies in various formats from disk or web or by specifying the URL.
  - the export of ontology can be done in several formats: RDF / XML, OWL / XML, OWL Functional, Manchester OWL.
- support for graphical visualization
  - offers visualizations in several forms with the possibility of exporting as a picture.
- support for reasoning
  - none
- support for queries
  - none
- usability
  - ☐ creating an ontology
    - in a first view, it is possible to fill in the name of ontology, prefix, namespaces, comment and annotation. In the annotation you can choose from a list the desired item and the language.
  - ☐ creating classes
    - the class name, equivalent classes, superclasses, disjoint classes and keys can be passed.
    - the application does not help you choose disjoint classes, equivalent classes or superclasses
    - although we have passed the superclass, the graphic link does not automatically appear
  - ☐ adding object properties
    - in the characteristic graphic style, if for classes we have boxes, for properties we have lines (association). We unify with a line the

two targeted classes and there is a view where we can complete the name of the property, the anotations, the property characteristics (functional ....), the equivalent properties, the superproprietals, the disjoint properties. in the same view, inverse (inverseOf) properties can be passed since it is a property between the same two classes.

- ☐ adding data properties
  - select box class + properties. you can fill in the name, type, annotations, if it is functional, equivalent properties, superpropriety, disjoint.
- ☐ adding individuals
  - you can fill in the name, type (choose from a list of already created classes), comment, annotation, difference or equivalence with other individuals
- ☐ importing an ontology
  - from the beginning, the program tells you to choose whether you want to visualize an existing ontology or create one. After selecting the import option, you can load an ontology from the disk. A graphical ontology visualization with the possibility to modify the elements appears.
- ☐ visualization
  - you can set the background, how to place the graphic elements in space
  - each box can be set to box type (rectangle, ellipse, octagon, etc.), background color, font, other effects (shadow, 3D). Changes can only apply to the selected item or to all items of the same type.
  - export can be done as image in various formats including SVG.

Fluent Editor

- ➢ hardware/software resources

- the supported operating system is only Windows including version 10.
- is available as a downloadable installation file (*.msi);
- in the case of version 3.6.10.28710, the archive has a size of 64.4 MB
- free licensing for individual, academics or small professional teams.

➢ interoperability

- compatibil with Protege 5.0; has the option to import or export to Protege;
- can import ontologies from a URL or from a local file. Free edition only allow the import of small ontologies with less than 200 lines of CNL statements.
- can export in format RDF/XML, OWL/RDF, as an attachment to mail or export to Ontorion (a distributed knowledge management system that uses CNL interface)

➢ support for graphical visualization

- offers visualizations in two forms with the possibility of exporting as a picture (*.jpg,*.svg).

➢ support for reasoning

- reasoning engine : HermiT

➢ support for queries

- SPARQL

➢ other facilities

- to extend the expressivity of OWL, this editor appeals to SWRL - Semantic Web Rule Language.

➢ Usability

- Fluent Editor uses Controlled Natural Language to create and model ontologies. The first thing to do before starting an ontology project is to learn and understand the CNL syntax and grammar.
- naming convention - if we were accustomed to ontologies with CamelCase notation for classes and mixedCase for attributes, in this case all identifiers are separated by dash, the class and property identifiers are written in small letters, individuals are written with the

first large letter separated by dash. All axioms appear in document view. The editor corrects you when entering any sentence that is grammatically or morphologically incorrect and helps you with suggestions during sentence writing[27]. On the right side you can see the taxonomy tree, something like the Protege class hierarchy.

- selecting a few statements, you can view their XML form.

☐ creating an ontology

- to complete the namespace, use the keyword 'Namespace :' followed by IRI.
- instead of prefixes, references are used with the keyword "References" and in statements the element of the other ontology is succeded by the reference prefix.

☐ creating classes

- to create a class it is used concept subsumption - one concept subsumes the other one if the set described by the first concept is a subset of the other one (Algorithm subclassOf Thing <=> Every algorithm is a thing).
- to express class equivalency it is used the keyword 'if-and-only-if-it'.
- the disjunction is expressed by negation, keyword "no".
- you can declare into a single expression, disjoint union of several classes with the keyword "if-and-only-if-it-either".

☐ adding object properties

- the properties of objects, called roles, are also added by a statement. There may be existential role restriction ('Every problem is-resolved-by an algorithm') and universal role restriction ('Every algorithm resolves nothing-but Problem').
- complex expresion
- intersectia
- uniunea
- cardinal restrictions can be expressed using the keyword (less than, more than, at-most, at-least, different-than).

- you can express transitivity, reflexivity ('Every problem has-sub-problem itself.'), irreflexivity (a kind of reflexivity negation), symmetry, role equivalence ('if-and-only-if');

☐ adding data properties

- to specify a datatype of data property you can specify into brackets the type of data literal (some value), integer (some integer value), double (some real value) boolean ('Every algorithm is-complete nothing-but (some boolean value)'), string (some string value), date (some datetime value).
- you can use the following keywords to specify a range of values (greater-than, lower-than,greater-or-equal-to,lower-or-equal-to, different-from, equal-to).

☐ adding individuals

- to specify individuals (here instances of concepts), a statement is sufficient ('Maze is a problem.'). For individuals, the name begins with a capital letter and without the expression 'every'.

☐ importing an ontology

- import can be made from disk. the editor translates the ontology axioms into the CNL and displays them as such and in taxonomy tree

☐ reasoning

- in the reasoner tab, we can fill in which element we are interested in. For example, we are interested in the a-i-algorithm. We write the question in that box 'Who-Or-What is a a-i-algorithm'. The editor simultaneously displays the answer to three variants of questions : <?> is a a-i-algorithm (individuals); every <?> is an a-i-algorithm (sublasses); every algorithm is a <?>(superclasses).

☐ querying

- you can query SPARQL using the rdf format, the prefix may be specified for ease of the query. The result can be copied to the clipboard from the resulting table right clicking over the

selected rows in the control, this content can be pasted to any spreadsheet software.[26]

- visualization
  - there is the possibility of a graphical visualization of classes, but the editor relies heavily on the existence of an R-plugin. Only two view variations are available (hierarchical layout and force directed layout). There are some options on the display data.
  - the visualization can be saved as *.jpg or *.svg.

- others
  - plugin for R-project

V. CONCLUSIONS

After studying the ontology editors, according to the criteria considered, some conclusions can be drawn:

- hardware / software requirements

Protege and OWLGrEd are Java-based and can be run on any platform that has JRE (Java Runtime Environment). Larger dimensions and memory requirements in the Protege case are explained by the addition of reasoning, quering, etc., although at visualizing of large ontologies are more prone to crash. Fluent Editor requires installation on a windows-based system. WebProtege on the other hand does not require installation, just a browser and a e-mail account.

- interoperability

Protege has most of the ontology serialization formats for import / export (Turtle, OBO, LaTeX, Jason-LD) and the procedure seems simpler. Fluent Editor is less centered on the variety of import / export formats. Instead, it has a plugin that allows to import / export ontologies to Protege. Also, thanks to another plugin, it can communicate data with R-project.

Graphical visualizations can be saved as images in Protege and OWLGrEd with a plus for the latter having the possibility to save in SVG format.

- support for graphical visualization

At this caracteristic, OWLGrEd wins detached due to the extensive customization capabilities of graphics. Some prefer to build the ontology in Protege and visualize it in OWLGrEd. Protege thanks to a plugin can provide a pretty suggestive graphical view.

- support for reasoning

Of the applications studied, only Protege and Fluent Editor has this feature, with an implicit reasoner (HermiT) and the possibility to install other reasoners. Hermit verifies the consistency of ontology, explains the inconsistencies found, and makes inferences based on existing data. Explanations in case of inconsistency or in case of inference are more detailed in Protege.

- suport for queries

Protege offers the possibility of making queries (DL and SPARQL queries).

Fluent Editor allows interrogations in SPARQL and CNL.

- usability

The editors studied were designed for different purposes, address different user categories, and have different characteristics. Some address beginners and emphasize the graphical side and ease of creating and modifying simple ontologies. Others address specialists, are built for more complex ontologies, have more facilities and have a wider range of applications.

For beginners and those who are familiar with UML diagrams and object-oriented programming that intends to make a simple ontology, OWLGrEd is the most appropriate editor. It is based on graphical diagrams in UML style, classes, object and data properties, individuals and class relationships are built through graphical elements. The emphasis on the graphic is also reflected in the view mode. In it you can set the colors, the font, the shapes, the spatial layout. An ontology can be created in Protege and imported into OWLGrEd to get a great visualization, especially for large ontologies. The resulting graph can then be saved in several formats including SVG for web pages. A Protege plugin also allows the creation of classes and relationships between them in graphic style. In Fluent Editor, you can not do anything unless you understand concepts and grammar, but after this stage things can be expressed much easier than in other editors.

For ontologies worked in team the ideal editor is webprotege. This, besides taking over many of the Protege facilities, has the advantage that ontology can be accessed online. It means that you can modify the ontology from many places, shared with other in the github style with rights set for each user. And Fluent Editor offers this opportunity of Collaborative editing of Ontologies by connecting to the Ontorion server, but it's only for a limited time

For specialists, Protege is the application of choice. Creating class hierarchy, especially using indentation, one class per row, is very simple and requires no specialized knowledge. As we move forward, more knowledge is needed, but the modeling possibilities grow. Firstly, it has a reasoner and can detect the inconsistencies in ontology. Secondly, inherited inferences can be integrated into ontology. Finally, interrogations can be made in DL and SPARQL. Protege can also be used to convert ontologies from one format to another, with a wider range of import / export options. with backing a community that supports it, has the advantage of a large number of plugins that makes it more versatile. OWLGrEd

also has a number of plugins, but it does not compare. Fluent Editor can easily express complex expressions, but things seem more organized and clearer in Protege.

As future intentions, I propose to find other evaluation criteria, to extend the demonstration ontology with various elements and axioms, and to the extent possible to encompass in the study also editors that are not freeware (trial or shareware).

Bibliography

[1] https://disenowebakus.net/en/web-evolution

[2] https://blockgeeks.com/guides/web-3-0/

[3] International Journal of Computersand Technology, November 2017, Concept and Dimensions of Web 4.0, Fernando AlmeidaAssociate Professor, Faculty of Engineering of Oporto University, INESC TEC, Porto, Portugal https://pdfs.semanticscholar.org/d38c/0b90e2501a34c8d8da9f985b079dea778d6b.pdf

[4] Berners-Lee, Tim; James Hendler and Ora Lassila (May 17, 2001). "The Semantic Web". Scientific American Magazine

[5] https://en.wikipedia.org/wiki/XML

[6] A Developer's Guide to the Semantic Web, Liyang Yu, Springer-Verlag Berlin Heidelberg 2011

[7] https://en.wikipedia.org/wiki/Simple_Knowledge_Organization_System

[8] https://en.wikipedia.org/wiki/SPARQL

[9] web09SemanticWeb-Ontologii-OWL.pdf

[10] https://en.wikipedia.org/wiki/Resource_Description_Framework

[11]web10SemanticWeb-Ontologii-OWL2-BazeDeCunostinte-LogicileDescrierii-Rationamente.pdf

[12] JSON for Linking Data https://json-ld.org/

[13] OBO Format, http://ontogenesis.knowledgeblog.org/245?kblog-transclude=2

[14] https://en.wikipedia.org/wiki/LaTeX

[15] web10SemanticWeb-Ontologii-OWL2-BazeDeCunostinte-LogicileDescrierii-Rationamente.pdf

[16] https://www.w3.org/OWL/

[17] https://en.wikipedia.org/wiki/SMA*

[18] https://en.wikipedia.org/wiki/Hill_climbing

[19] Artificial Intelligence A Modern Approach *Third Edition* Stuart J. Russell and Peter Norvig 2010

[20] https://en.wikipedia.org/wiki/Hill_climbing

[21] https://protege.stanford.ed

[22] http://owlgred.lumii.lv

[23] http://docs.cognitum.eu/FluentEditor

[24] https://www.slideshare.net/Cognitum/introduction-to-ontology-engineering-with-fluent-editor-2014

[25] www.cognitum.eu/

[26] http://docs.cognitum.eu/FluentEditor/