WORK PLATFORMS WITH ONTOLOGIES IN TOURISM

Eugenia IANCU
Stefan cel Mare University of Suceava, 720229, Romania
eiancu@seap.usv.ro
Aurel BURCIU
Stefan cel Mare University of Suceava, 720229, Romania
aurelb@usv.ro

Abstract
The emergence of the Internet has led to an explosion of web applications. The World Wide Web (www) has a huge and permanent influence on our lives. Economy, industry, education, health, public administration are components of our lives that have not been penetrated by the World Wide Web. The reason for this omnipresence lies mainly in the nature of the web, characterized by the global and permanent availability but also by the homogenous access to the information distributed globally produced by individuals in the form of web pages. Initially, the web was designed as a purely informational environment, currently evolving into an application environment. Semantic web technologies and ontologies respectively are used in the development of e-learning systems in order to represent models and manage learning resources in a more explicit and efficient way. Intelligent training techniques such as personalized learning and adaptive systems are a current concern in the context of diversity of user profiles. The present work aims to construct an ontology using OWL (Ontology Web Language) and RDF (Resource Description Framework) in the field of tourism and a platform for working with ontologies. The platform that will be represented by a web application will allow users to expand an ontology by adding new concepts such as classes, subclasses, individuals, properties, but also visualize the ontology in the form of a knowledge graph. Also, users will be able to interact with the ontology by querying it using the SPARQL language.

Key words: *classes, concept, instance, ontologies, patterns, query*

JEL Classification: *L83, M15*

I. INTRODUCTION

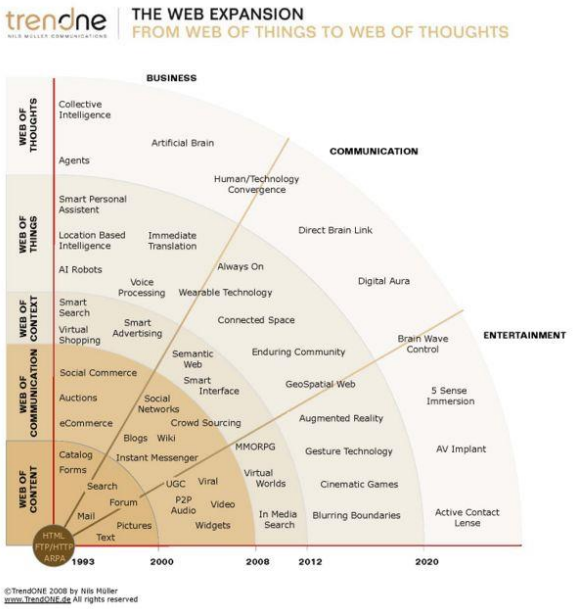
Today's web applications are fast and represent complex software systems that offer interactive and customizable services accessible through different devices; they provide the ability to perform transactions between users and usually store the data in a database. The distinguishing element of web applications, compared to traditional software applications, is how the web is used: for example, its technologies and standards are used as a development platform and as a user platform at the same time. The World Wide Web (WWW) has evolved since its inception through several stages. The W3C consortium considers these steps as (7):

1. Web 0.0 - Developing the Internet
2. Web 1.0 - The Internet prior to 1999 was considered by experts as a read-only Internet. The role of a user was just to read the information presented to him. There was no active communication or information flow from producer to consumer. Virtually the first stage of the Web allowed users to search and read information only.
3. Web 2.0 - The writing and participating web. Due to the lack of active interaction of Internet users with the Internet, Web 2.0 was born. The year 1999 marked the beginning of the "Read-Write-Publish" era, which was marked by the appearance of "Live Journal" (April 1999) and "Blogger" (August 1999). Tim Berners Lee said about Web 2.0 that it is a "read-write Web" and that users now have the ability to contribute to the content of web pages and interact with other users. This stage changed the drama of the Internet and the world and was a response for users who wanted to be more involved in what information was available to them. Highlights of this stage include Twitter, YouTube, Facebook, Flickr, which have changed the concept of human interaction.
4. Web 3.0 - The semantic executing web "The Semantic Web is an extension of the current Web that allows the formal description of existing resources on the Internet (web pages, text and multimedia documents, databases, services, etc."). Tim Berners Lee said that this web site is of the "write-read-execute" type. The bases that will form web 3.0 will be semantic markup and web service. Semantic markup refers to the lack of communication between human Internet users and computers. One of the most The big challenges are finding a way to represent the data so that it is understood by human users and software agencies. By combining a semantic markup

with a web service, Web 3.0 promises the potential of applications that are able to speak directly and with each other and look for information using simplified interfaces. If Web 2.0 meant too much information, Web 3.0 will bring you to the center of the stage concepts such as personalized search, information deduction, 3D Web.

5. Web 4.0 - Mobile Web This concept involves a new version of the Internet based on what we already have, but adapted to mobile devices. The purpose of Web 4.0 is to connect all mobile devices in both the real world and the virtual world.

6. Web 5.0 - Next Web. Tim Berners Lee says Web 5.0 is "Open + Linked + Intelligent Web = Emotional Web" Web 5.0 is still in development, but signals show that this Web will be an Internet that communicates with us as a kind of personal, access assistant , is also called "symbiotic Web". This web site will be of type "read-write-execution-concurrency". At the center of this concept will be the emotional interaction between people and computers. This interchange will become a daily habit due to the evolution of neuro-technology. At this moment, the Internet is neutral because it does not perceive the emotions and feelings of users. A representation of the evolution of the Internet is depicted in Figure 1.



Source: https://flatworldbusiness.wordpress.com/flat-education/previously/web-1-0-vs-web-2-0-vs-web-3-0-a-bird-eye-on-the- definition /

Figure 1 Evolution of the Internet

The Internet has evolved a lot over the last ten years and as such the possibilities of working with. With the evolution of the Internet and the working methods have become more and more complex. At present we are talking about ontologies, possibilities of working with ontologies, the coverage area with these ontologies. Considering the most cited definitions in the field, that of Gruber, "an ontology is a specification of the conceptualization of a domain" (Gruber, 1993). An ontology contains predicates, the semantics of concepts and terms, but also how they relate to each other. Other more recent definitions of ontologies speak of these as a "knowledge graph" (Noy and McGuinness, 2001). A knowledge graph is the sum of an ontology and a set of concrete instances of the classes.

The proposed application is part of the applications that work with ontologies and is useful for viewing, querying and extending ontologies. The first objective of the project was the construction of an ontology using the RDF standard and the OWL language. The domain chosen for the construction of the ontology was that of tourism. An ontology is a catalog of the concepts existing in a domain. Graphical ontology visualization.

One of the objectives of the application was to offer users the opportunity to graphically visualize a created or existing ontology. In order to do this, the user will need the json file that he will upload to the application and based on which the ontology graph will be generated. The user will be able to choose for which concepts he wants to generate / visualize the relations of connection. This will use a graphing technology provided by JavaScript called D3.js. By default D3 has implementation for trees. As the ontology is a knowledge graph, an algorithm will be implemented that will allow to graphically represent the concepts in the ontology. Java frameworks and ontologies Because the application has logic written in Java, it was necessary to find a way to work with .owl files in Java. There are several Java frameworks that offer this. These include (11,12,13):

• JAOB (Java Architecture for OWL Binding)
• OWL API - is a reference interface for creating, manipulating and serializing OWL ontology)
• JENA Framework - is an open source framework that allows you to build Linked Data, Semantic Web applications and work with ontologies.

For this application, the Jena framework is used because it offers an API for extracting and writing data in a graph. Graphs are represented as abstract patterns that can source data from files, from the database, from URLs, or a combination of them. Also, models can be queried using SPARQL. Jena is similar to Sesame, but unlike this it offers support for OWL. It also has internal reasoning engines such as Pellet Reasoner (which is a Java representation of the OWL-DL reasoning engine).

II. CREATION OF REST WEB SERVICES

In order to provide users with functions for handling and querying ontologies, a number of REST services have been created and implemented. The services will interact with the interface through AJAX calls. The

services created are the following:
• Service that allows to query an ontology in the form of a SELECT type query;
• Service that allows querying an ontology in the form of an ASK query;
• Service that allows querying an ontology in the form of a CONSTRUCT query;
• Service that allows the addition of a specific court for a given concept;
• Service that allows adding a new concept;
• Service that allows the addition of a new property;
• Service that allows adding a property to a particular concept.
REST or representational state transfer is an architectural style that consists of a set of components, connectors and a distributed system and that focuses on how the components communicate with each other. The purpose of REST is to bring performance, simplicity, scalability, simplicity, visibility, portability to web services. There are some architectural constraints that must be met by any REST services and I will detail in the following:
• *Server Client* - a uniform interface separates server clients. For example, customers are not interested in how data is stored (this only concerns servers), so it increases the portability of the client code. Servers are not interested in interface or status, so they can be simpler and more scalable. Servers and clients can be replaced and developed separately from each other, as long as the connection between them (the interface) is not affected.
• *Cacheable* - responses must be defined (explicitly or implicitly) as cacheable or not, in order to prevent clients from using inaccurate data in response to the following requests to the server. This can eliminate some of the interactions between client and server, improving performance and scalability.
• *Stateles*s - communication between client and server is constrained by the fact that no context from the client should be stored on the server between successive requests from the client. The session status can be transferred from the server to a specific service such as a database to maintain a persistent state for a certain period. The customer starts sending requests when they are ready to move to a new state.
• *Layered system* - the client cannot specify whether it is connected to the central server or to an intermediate server. Proxy servers can improve system scalability by enabling load balancing (this means that requests from clients are distributed to less loaded servers) or by providing copies of Chace. This can also lead to improved security policies.
• *Uniform interface* - the uniform interface is a fundamental constraint of any REST service. It is the one that simplifies and decouples the architecture, so that the client and the server are completely independent.
For an interface to be uniform it must meet the following conditions:
• Identification of resources - individual resources are defined using URIs. Resources are separate from the representation that is returned to the client. For example, the server can send data in HTML, XML or JSON format, even if they are not among the server's internal representations.
• *Manipulation of resources through these representations* - when a client holds the representation of a resource, it has sufficient information to modify or delete the resource.
• *Self-descriptive messages* - each message must include information on how the message should be processed.
Web services that adhere to REST specifications are called RESTful APIs and are defined under the following aspects:
1. Have a basic URL;
2. A data type: Usually it is JSON, but it can be any media type on the Internet
3. With the standard HTTP method: GET, POST, PUT, DELETE, OPTIONS

III. CREATING THE WEB INTERFACE

One of the objectives of the application was the implementation of an interface using the web services described in the previous point. The purpose of this interface is to provide users with the tools they need to be able to extend, manipulate, view and query an ontology in OWL format.The application must respond to a series of requirements that have the purpose of functioning properly:

Response time - being a web application, the acceptable response time is in the order of milliseconds. Given that the application has a graphical part of the ontology using the javascript D3 library, the generation and loading time of the graph should be around tens of milliseconds. Also, the data transmission from the interface to the server, as well as vice versa, must be in the order of tens of milliseconds.

Responsibility - the user interface should be a "responsive" one. The concept of “responsive” can be defined as the property of the interface to respond promptly to the user's requests, without blocking or letting the user wait. We can look at the response time as the time difference from when a user action generates an event and until a response is received as a consequence of the event.

Scalability - In the software industry, scalability is defined as the property of a software application / system to properly support a larger volume of loading or to allow its expansion or extension. Thus the application will have to behave the same for a small number of users, but also for a large number. Also, being a modular application that is based on REST web services, it can be easily expanded to add new features.

Reliability - the concept of reliability is translated by the tolerance to errors and the degree of recovery after the production of errors. Thus, the application must support the possibility for users to enter the data incorrectly, otherwise there must be a validation of the data being entered. Data validation can be performed both on the interface side and on the server side. Also, data validation also helps prevent data attacks such as XSS Attacks. Because owl files will be able to be uploaded, there will also need to be validation functions to validate what kind of files users can upload. *Testing* - The application will have to work properly in the scenario in which the data is entered correctly. Thus, in this scenario, the results obtained are those of expectation, but also in the scenario where there are errors in the data entry. In this case, suggestive error messages will appear and the data processing will not continue.

IV. SPARQL

It is a language that allows us to write queries for ontologies. It resembles SQL and uses the same triplet structure. Next I will try to present some of the features of SPARQL and how it can be used. The SPARQL language is based on the triplet structure. The structure of triplets is similar to that of the RDF (subject - predicate - object) with the only difference the subject, predicate and object can be variable instead of RDF terms. These triples are combined with a graph model where exact matching is required to find the results.

Similar to the namespace mechanism in RDF, SPARQL lets us define prefixes for namespaces and use them in queries to make them shorter and more readable.

SPARQL does not explicitly support RDFS semantics. So the query result depends on whether the system supports RDFS semantics. If yes, then the result of this query will include all instances of the subclasses of that class. If not, then only those instances that explicitly have the class type will be returned.

Using triple select-from-where Like SQL, queries in SPARQL have a form like SELECT-FROM-WHERE. SELECT specifies the number and order of the data to be returned, FROM specifies the source to be used for querying, and WHERE imposes constraints on possible solutions in the form of Boolean constraints or graph templates. The FROM clause is optional: when not specified, we can suppose that we query the knowledge base of a particular system.

For example, to get all the phone numbers of staff members, we can write a query like:

```
SELECT ?x ?y
WHERE
{ ?x uni:phone ?y . }
```

Here? x and? y are variable, and? x uni: phone? y represents a source-property-value pattern. More complicated patterns can be created to obtain complex information from queries. For example, to return all students and their phone numbers, we can write a query like:

```
SELECT ?x ?y
WHERE
{ ?x rdf:type uni:Student ; uni:phone ?y . }
```

Queries in SPARQL

Here the clause? x rdf: type uni: Student collects all instances of the Student class and assigns the result of the variable? x. The second part collects all the triples with the predicate phone. This query has an implicit join in which we restrict the second part only tox. For this the syntax is used; which indicates that the next triple will have the same subject as the previous one. So, the above query is equivalent to this:

```
SELECT ?x ?y
WHERE
{ ?x rdf:type uni:\student . ?x uni:phone ?y . }
```

The following demonstrates an explicit join of a query that returns the name of all the disciplines that the student has passed with ID 646352:

```
SELECT ?n
WHERE
{ ?x rdf:type uni:Discipline ; uni:isTaughtBy
:646352 . ?c uni:name ?n .
FILTER (?c = ?x).
}
```

SPARQL uses a FILTER condition to indicate true or false constraints. Above, constraint is the explicit join of variables? c and? x by using the equality operator (=).

Optional Patterns

All the above patterns are required. Either the knowledge base matches the pattern in which case it returns a result, or not, in which the query will return nothing. However, sometimes we want to be more flexible. Let's look at the following example:

```
<uni:student rdf:about="646352">
<uni:name>Grigoras Claudiu</uni:name>
</uni:student>
<uni:student rdf:about="646318">
<uni:name>David Bores</uni:name>
<uni:email>david@yahoo.com</uni:email>
</uni:student>
```

This excerpt contains information about two students. For one of the students, only the name is displayed, for the other, the e-mail address is also displayed. Now we want to write a query that will return the names of all students and their email addresses.

```
SELECT ?name ?email
WHERE
{
?x rdf:type uni:Student ;
uni:name ?name ;
uni:email ?email .
```

}

The result of this query will be? Name? Email David Bores david@yahoo.com. So, although Grigoras Claudiu is listed as a student, the question will not return his name because he does not have an e-mail address and does not fit the pattern. As a solution, we can adapt the query to use an optional pattern:

SELECT ?name ?email
WHERE
{
?x rdf:type uni:Student ; uni:name ?name
.
OPTIONAL { ?x uni:email ?email }
}

This query: "returns all students' names and if their email address is also known" and the result will look like this:? Name? Email Grigoras Claudiu David Bores david@yahoo.com.

In this part of the article I tried as much as possible to render the basic functionalities of the SPARQL language used in the application. The ontology I developed is a small ontology on

-Tbox: terminal box - is that part of the ontology that contains the concepts defined by it. Here are the entities: ontology classes, object properties, properties

-Abox: assertion box - represents that part of the ontology that contains the instances of the ontology concepts.

An ontology brings together concepts with individual named courts. The ontology structure we developed (with the help of Protege) is shown below. The classes with the corresponding subclasses are the following: **Accomodation (BedAndBreakfast, BudgetAccomodation, Campground, CountryHouse, Hostel, Hotel -> LuxuryHotel, Inn, Pension), Activity (Adventure – Alpinism, BunjeeJumping, CliffJumping, Diving (CaveDiving, ScubaDiving), ExtremeSport, Mountainbiking, Rafting, CulturalActivity – Cinema, Opera, Theater, Event – Concert, Exposition, Relaxation – Beach, Dance, Shopping, Sunbathing, ExperienceBath, Sport – AirSport, Alpinism, Bowling, Boxing, ExtremeSport, Golf, Hiking, HorseRiding, MindSport, Mountainbiking, Tennis, WaterSport, WinterSports), BlogPost, Destination (Country – City (Capital), County, RuralArea – Village, UrbanArea), EatingAndDrinking (Bakery, CashAndCarry, HerbsAndSpice, Market - Supermarket, OrganicHealthAndKosherFood – Restaurant (ChineseRestaurant, ItalianRestaurant, TraditionalRomanianRestaurant)) Season (Spring, Summer, Autumn, Winter), Transport (Air – Airport, Helipad, RoadAndRail, Water – Boat Station, FerryStation, TransportAccessPoint – Bus Stop, MetroStation, Parking, RailwayStation, TaxiStation, TramStation, UndergroundEntrance).**

Object properties are: **axNumber, has Author, hasAddress, hasContent, hasDescription, hasImage, hasLat, hasLong, hasMaxBedNo, hasName, hasEnglishName, hasRoomNo, hasShortName, hasSource, hasStars, hasTag, hasTitle, hasWebpage, importDate, link (wikipedia, dbpedia, freebase, wikitravel), phoneNumber, populationOf, publicationDate, refferedBy, wikipediaContent, wikitravelContent**

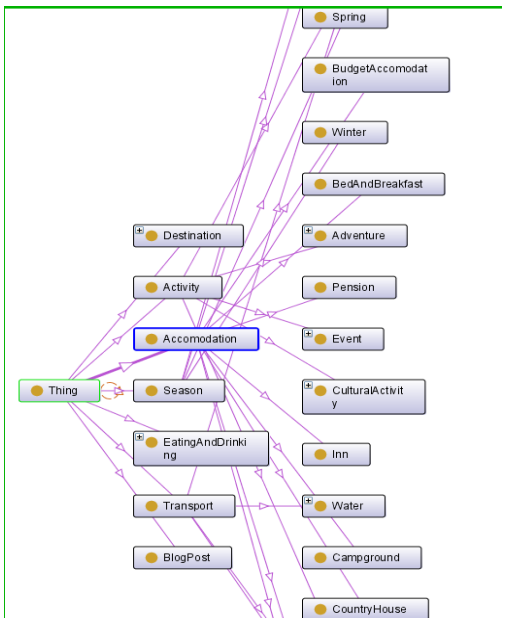


Figure 2 The ontology structure

The application is a web platform that allows working with ontologies. Thus the users have the following functionalities:

• I can upload a json file to the application. The .json file is a conversion of the owl file. The transformation will be done with the converter and that comes with the application;

• Based on the uploaded json file you can generate a graph for the loaded ontology;

• I can add instances for specific classes in ontology;

• I can add a new property within the ontology;

• I can add a new property to a concept within the ontology;

• I can export all properties within the ontology in an XML format;

• I can write queries using SPARQL to get ontology information. The queries that can be written are of three types: CONSTRUCT type, ASK type and SELECT type. Examples for these types of queries and how they can be used will be presented in the testing part of the application.

V. COMPONENTS OF THE APPLICATION

The application has three major components:

• The ontology created

• Back-end part (application logic)

• The user interface part.

Next I will present implementation details about the two back-end components and the interface one. The back-end part contains the logic of the application and is written in Java as REST web services. More details on what a REST web service is and how it works have been presented previously. Basically the application has five REST services that communicate with the interface. Each service performs a particular functionality and is mapped to an HTTP method (for all services in the application the method is POST) and has a URL mapped to the code that can be accessed. These services are mapped to Java classes as follows:

• AddIndividualsService - allows the user to add an instance to an ontology class. It receives from the interface the names of the ontology, its namespace, the instance and the class to which it belongs and assigns in the ontology the instance of the class.

• AddPropertyToIndividualService - allows the user to add a property to a court. It receives from the interface the ontology names, its namespace, the name and value of the property and the instance, and adds in the ontology the property for that instance.

• CreateObjectPropertyService - allows the user to add a property within the ontology. It receives from the interface the ontology name, its namespace, the name, the domain and the value range of the property and writes the property in the ontology.

• OntologyUtilsService is a two-way utility that allows you to open and write an ontology.

• QueryingOntologyService - allows the user to write queries of three types: SELECT, CONSTRUCT, ASK. The service has three methods for the three types of queries. Each method receives the ontology name, namepace, IRI and the ontology prefix if defined, as well as the query to execute and executes the query, returning the results obtained in the interface.

Figure 3 shows a structure with a Service type class.

```
@Service
public class CreateObjectPropertyService {

    public void createObjectProperty(String ontologyName, String namespace, String propertyName, String domain,
            String range) {
        final OntModel model = OntologyUtilsService.openOntology(ontologyName, namespace);
        ObjectProperty objectProperty = model.createObjectProperty(namespace + propertyName, true);
        Resource res1;
        if (domain != null) {
            res1 = model.getResource(namespace + domain);
            objectProperty.addDomain(res1);
        }
        Resource res2;
        if (range != null) {
            res2 = model.getResource(namespace + range);
            objectProperty.addRange(res2);
        }
        OntologyUtilsService.writeIntoOntology(ontologyName, namespace);
    }
}
```

Figure 3 Service class

Each service is mapped to a controller class. A controller is an application class in which the processing of a request from the interface begins. The structure of a controller type class is detailed in the Application Structure sub-chapter.

The application interface is built using HTML5, CSS, JavaScript, D3.js, Angular.js and AJAX technologies. HTML5 is used to build the interface. The interface will look like in Figure 4 below:

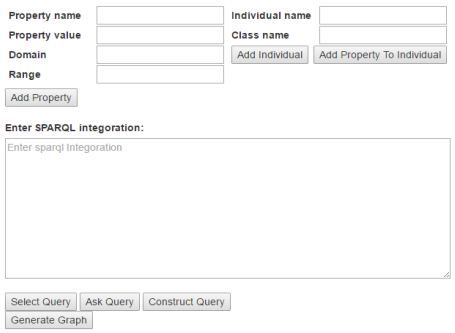


Figure 4 Web application interface

CSS is used to style the HTML elements of the interface. Javascript along with D3.js and Angular.js framework will be used to create the graphical visualization of the ontology. Ajax (Asynchronous JavaScript and XML) is a technology used to create interactive web applications. Ajax is used to exchange amounts of data with the server.

VI. RESULTS

Application validation refers to data validation. This is done on both the interface and server side. The data entered by the user is validated on the interface side. The same data is also validated on the server side. If the data is not conclusive with the validation, the user will be informed by the corresponding error messages.

Next, I will present three test scenarios: in the first scenario, a new instance will be added, in the second one a few SPARQL queries will be written to obtain relevant information about concepts, and in the third one the ontology graph will be created. .

The first test scenario: Adding a court for a concept. In the following, I will reproduce the steps taken with the appropriate images.

A restaurant named "Zamca" will be added in the "TraditionalRomanianRestaurant" category. In this case, the court is "Zamca" and the class name is "TraditionalRomanianRestaurant". Figure 5 shows the completion of the required fields.

1. The user fills in the interface name of the court and the name of the class to which the court is added.

2. The user clicks on the Add Individual button and the court is added.

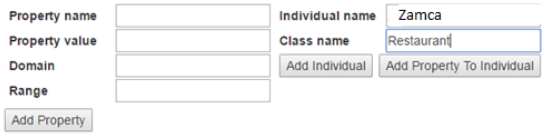


Figure 5 Fill in the fields when launching the application

3. To observe the new court, you can use Protect or write a SPARQL query that returns all the restaurants in the class: "TraditionalRomanianRestaurant".

The second test scenario: Writing SPARQL queries. In this test scenario we will write two SPARQL queries.

1. The user will write the first query that will return all the restaurants in the TraditionalRomanianRestaurant category together with their number of stars. The query looks like Figure 6. Because it is a Select query, you will press the Select Query button.

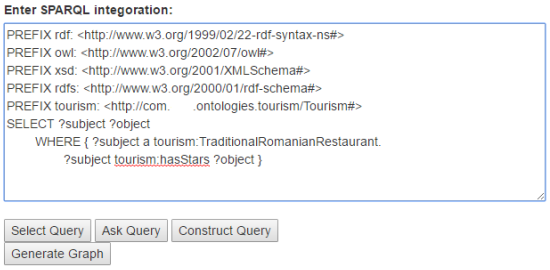


Figure 6 SPARQL query

Third test scenario: Generation of the ontology graph. To get the graph, the user will perform the following steps:

1. Press the Generate Graph button. After pressing it, a window will appear as in Figure 7:

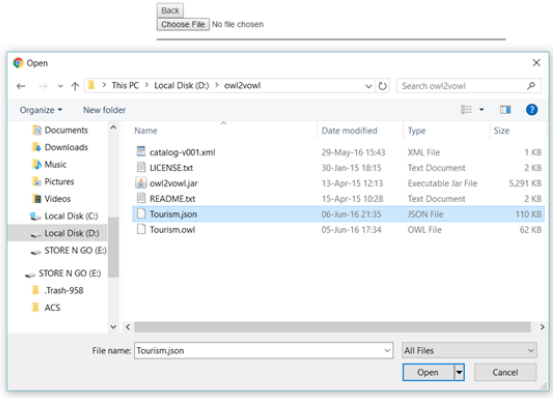


Figure 7 Generation of a graph

2. The user will select the json file corresponding to the owl file (in the image above it is Tourism.json) and click Open. The graph will be generated as in Figure 8

Figure 8 The graph generated by the application

Initially, the graph will have only the Thing object. In order to be able to extend a concept (view its subclasses) or to be able to narrow a concept (view its subclasses), the user will have to click on that concept. The application was developed using the Eclipse STS IDE (Eclipse Spring Tool Suite), Java 8, Tomcat 8, and Node.js development environment. Java 8 (JDK 1.8) or later, a Tomcat server (version 8 or later) and a Node.js server for Angula are required to install the application and run it locally.

VII. CONCLUSION

The purpose of this work was to build an ontology in the field of tourism and to create a platform that allows the visualization, extension and interrogation of the created ontology. The three main objectives resulting from the functional requirements w

• Visualization: graphical ontology creation using D3.js and Angular.js;

• Expansion: the possibility to add instances, properties, new concepts by using the JENA framework that allows working with ontologies in Java;

• Query: Adding a SPARQL + JENA module that allows the user to write queries to get information about ontology concepts.

The application is error-tolerant and alerts the user with appropriate messages when an error or operation that could not be completed has occurred; being a modular application with a 3 tier architecture in which the interface and logic are clearly delimited, the modification or addition of new functionalities is very easy.

A further development that can be brought to the application is the further extension of the ontology by automating this process. One way to do this is by using crawlers to parse pages of information (such as wikipedia, dbpedia, wikitravel, etc.), parse data, and save them in the ontology.

Another development we are considering is the creation of a system for recommending tourist locations (points of interest, hotels, restaurants, etc.) based on the

opinions of other users. This could be achieved through opinions expressed on travel blogs or specific sites. You can also make a version on mobile devices. The application being made in Java and based on RESTful web services, its transformation into a mobile application only involves the development of an interface adapted to mobile devices. This would result in increased portability and number of users.

Although a conclusion may review the main points of the paper, do not replicate the abstract as the conclusion. A conclusion might elaborate on the importance of the work or suggest applications and extensions. Make sure that the whole text of your paper observes the textual arrangement on this page.

VIII. REFERENCES

1. Corcho, O., Fernández-López, M., Gómez-Pérez, A., (2003), *Methodologies, tools and languages for building ontologies: where is their meeting point?* Data & Knowledge Engineering, 46 (1), July, pp. 41 - 64 , Elsevier Science, Amsterdam. Fikes
2. Dean Allemang, Jim Hendler, (2008) - *Semantic Web for the Working Ontologist – Effective modelling in RDFS and OWL*, Morgan Kaufmann Publishers
3. Gruber T.R. (1993) - *A Translation Approach to Portable Ontology Specifications*, Standford, California
4. Mari Carmen Suarez-Figueroa, Asuncion Gomez Perez, Enrico Motta, Aldo Gangemi, (2012) - *Ontology Engineering in a Networked World*, Springer
5. Noy, N.F. and McGuinness, D.L. (2001) *Ontology Development 101: A Guide to Creating Your First Ontology. Stan- ford Knowledge Systems Laboratory Technical Report KSL-01-05 and Stanford Medical Informatics Technical Report SMI-2001-0880*, 1-25
6. Shelley Powers, (2003) - *Practical RDF*, Sebastopol, O'Reilly & Associates
7. https://flatworldbusiness.wordpress.com/flat-education/previously/web-1-0-vs-web-2-0-vs-web-3-0-a-bird-eye-on-the-definition/
8. http://www.topicmaps.org/
9. http://andrei.clubcisco.ro/cursuri/f/f-sym/4ioc/labs/RDF_OWL.pdf
10. https://jena.apache.org/documentation/ontology/
11. http://wiki.yoshtec.com/jaob
12. http://owlapi.sourceforge.net/documentation.html