A Question Answering System on Domain Specific Knowledge with Semantic Web Support

Borut Gorenjak, Marko Ferme, Milan Ojsteršek

***Abstract*—**In today's world the majority of information is accessible via the World Wide Web. A common way to access this information is through information retrieval applications like web search engines. We already know that web search engines flood their users with enormous amount of data from which they cannot figure out the essential and most important information. These disadvantages can be reduced with question answering systems. The basic idea of question answering systems is to be able to provide answers to a specific question written in natural language. The main goal of question answering systems is to find a specific answer. This paper presents an architecture of our ontology-driven system that uses semantic description of the processes, databases and web services for question answering system in the Slovenian language.

***Keywords*—**Ontology, Natural language processing, Question answering system, Semantic web, Web ontology language, Web services, Semantic web.

I. Introduction

Nowadays the internet is becoming a huge dump of documents, links and all other sorts of information. Most common possibilities to explore this information are information retrieval applications such as web search engines [7]. We already know that web search engines flood their users with enormous amount of data from which they cannot figure out the essential and most important information.

Despite the fact that search engines are doing an excellent job, they still return too much inaccurate information. The solution to this problem can be found in the form of question answering systems, where the user gives a question in natural language, similarly to talking with another person. The answer is the exact information instead of a list of possible results. These answers can be retrieved from domain-specific knowledge corpuses or other external resources like web services.

This article is segmented into eight chapters. The following chapter describes ontologies and semantic description of domain-specific knowledge. The third chapter describes the process used for ontology mapping to the relational database. The following chapter interprets the use of question templates. The fifth chapter describes integration of external knowledge resources. This chapter also explains semantic description of web services. The sixth chapter reveals the importance of user behavior. The seventh chapter describes the architecture and processes of our question answering system. The eighth chapter presents the complexity of application user interface. Chapter nine concludes with the summary and suggestions for our future work.

II. Semantic Description of Domain Specific Knowledge

The majority of information available on the web is suitable for human use. This is the main reason why computer applications have a problem understanding this data [2].

Fortunately, this problem can be solved by using the semantic web. Semantic web is an extension of the World Wide Web. As the name itself suggests, the purpose of the semantic web is to precisely define unambiguous computer understandable metadata, thus enabling computers and people to work in cooperation [4]. The main purpose of the Semantic Web is driving the evolution of the current Web by allowing users to use it to its full potential thus allowing users to find, share, and combine information more easily. The key element is that the application in context will try to determine the meaning of the text or other data and then create connections for the user. One of the most important components of the semantic web are ontologies which can significantly enhance understanding and description of information.

Ontologies are one of the main approaches used in the scope of knowledge management and artificial intelligence to solve questions related to semantics, with current relevance in the semantic web. They describe an abstract model of a domain by defining a set of concepts, their taxonomy, interrelation and the rules that govern these concepts in a way that can be interpreted by machines. Contemporary ontologies share many structural similarities, regardless of the language in which they are expressed. Most ontologies describe individuals (instances), classes (concepts), attributes, and relations [1].

Web Ontology Language (OWL) is a formal knowledge representation language for authoring ontologies. OWL makes the language intuitive for humans and to have sufficient expressive power to describe machine-readable content needed to support semantic web applications [12]. OWL satisfies the semantic web's requirements of providing minimal investment of human producers and consumers and supporting software

requirements for a language with explicit semantics.

The resulting OWL language is based on W3C standards and provides producers with information representation features to define their own ontologies and to extend others' ontologies. It supports expressive statements in a manner that supports scalability. OWL builds on XML and allows users to provide machine-readable semantic annotations for specific communities of interest.

OWL is used to make statements, called assertions, about classes, properties and individuals. Assertions can be stated in a single ontology or in a combination of multiple joined ontologies.

While the current web focuses on supporting humans reading text, its infrastructure provides opportunity for more sophisticated applications. One objective of OWL's developers was to provide layering of language features. Figure 1 presents one layered conceptual view of the semantic web. The layers shown are not true layers in sense of networking models but illustrate rough dependencies. Each layer depends on the layer beneath and uses their features to provide its capability. The figure shows that the top layer, the implementation layer, provides specific applications. In the layer below, the logical layer, OWL supports formal semantics and reasoning. Below OWL, the Resource Description Framework (RDF) Schema (RDFS) language supports ontological primitive layers. RDF supports the basic relational language layer through its simple data model and syntax for making statements. RDF is serialized using RDF/XML. XML and XML Schema data types support the transport/syntax layer and Uniform resource Identifiers (URIs) and namespaces support the symbolic / reference layer.

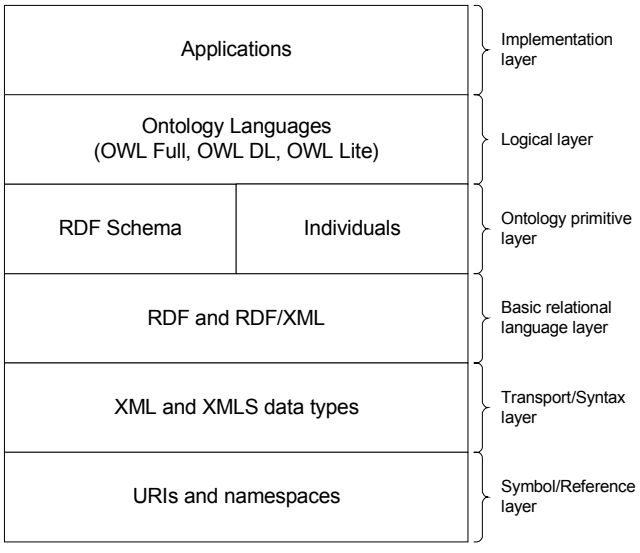

Fig. 1 Semantic web's layered architecture

As demonstrated, RDF(S) and OWL are useful languages for representing ontologies and metadata on the semantic web [13]. However, once this metadata has been published, query languages are required to make full use of it. SPARQL Protocol and RDF Query Language (SPARQL) aims to satisfy this goal and provides, as the name states, both a query language and protocol for RDF data on the semantic web. SPARQL can be thought of as the SQL of the semantic web and offers powerful means to query RDF triples and graphs. Trying to use semantic web without SPARQL is like trying to use a relational database without SQL. SPARQL makes it possible to query information from databases and other diverse sources across the web.

A RDF data is represented as a graph. SPARQL is therefore a graph-querying language, which means that the approach is different than SQL where people usually deal with tables and rows. Moreover, it provides extensibility within the query patterns (based on the RDF graph model itself) and therefore advanced querying capabilities based on this graph representation.

SPARQL can be used to query independent RDF files as well as sets of RDF files, either loaded in memory by the SPARQL query engine or through the use of a SPARQL-compliant triple store. Therefore, there is currently a need to know which files must be queried before running a query. This can be an issue in some cases and can be considered as a hurdle to overcome.

SPARQL offers four query forms that can be used to run different types of queries:

- SELECT, used to retrieve information based on a particular pattern,
- CONSTRUCT, used to create RDF graph based on RDF input and that can be used as a translation service for RDF data between different ontologies,
- ASK, used to identify if a particular query pattern can be matched on the queried RDF graph,
- DESCRIBE, used to identify all triples related to the particular object that must be described.

III. Ontology Mapping to the Relational Database

As we describe in the previous chapter, we needed a way to formalize ontologies. At the beginning we chose OWL for our knowledge representation [11]. We have been using Protégé, a free, open source ontology editor and knowledge-base framework [3]. It is a great tool for creating semantic web content, but we were concerned with its suitability for our end users. Our users are ordinary people, who do not know anything about ontologies and semantic web. OWL contains much more than we needed for our system. We also needed support for phrases and their synonyms.

All this led us to the conclusion that we have to build our own ontology representation. We took the idea of OWL, which has been reduced with irrelevant elements. We added Domain, Process, Phrases and Synonyms to our solution. We also added the semantic description of methods and parameters, which will be described in detail in the fifth chapter. Our ontology mapping to relational database is shown in fig.1.

All elements in figure 1 are presented as tables in relational

database. A domain is used to narrow information to a specific domain. It can be nested in other sub-domains and combines multiple processes for a specific domain. Domain can be nested in other sub-domains and combines multiple processes for a specific domain. We specify a domain with the title and corresponding description. A process is supported by certain knowledge, which is represented by classes. A class is the focus element of ontology. Classes describe concepts in the domain. Therefore every class needs its own properties. A property has a name, a description and a data type. We used only basic data types (integer, double, string, date …). We optionally extended data types with regular expressions. We added this feature for faster and easier searching by property instances. With such regular expressions we can accurately determine word phrases. Property instances are the most important part of ontology for us in term of detecting entities of question templates.

Word phrases and synonyms are usually not part of the ontology, but we have deliberately used them for our ontology. Every single word we are using in our ontology (name, title, description, instance values) is stored as word phrase in a special database table. Every word phrase is linked to its original element via synonym and could have one or more synonyms. This approach also determines ontology instances.

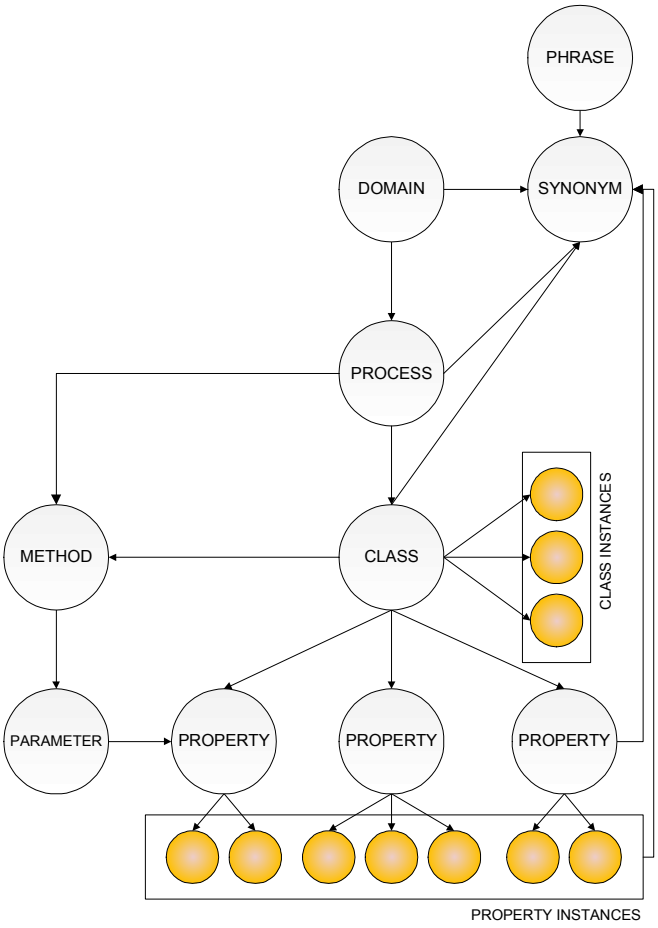

Fig. 2 Ontology mapping to relational database

While our question answering system supports external resources, methods and parameters are also included in our ontology representation. Methods are part of different processes. For easier and more accurate parameter determining, a method is also associated with a class in which they collect data for the parameters. While we are dealing with three different types of external resources (web service, DLL library and database stored procedure), we have to describe them exactly. We have a special attribute in a database table where we describe an external resource. Information that describes external resources is as follows:

- *Web service* – web service URL address, service name, method name, parameters.
- *DLL library* – DLL binary data, class name, method name, parameters.
- *Stored procedure* – connection string, command name, parameters.

Parameters in these descriptions are only pointers to the real description of the parameter. A parameter is stored in a special database table and is further linked to the class property. Every method represents parameters as a special class with properties. We already mention that properties are extended with regular expression. In situations when we need to call external method with parameters of different data type, it is very convenient to verify data with regular expressions.

It is important that the question answering system is able to provide answers as fast as possible. This is the main reason, why we built the whole idea as a relational database solution. We believe that a relational database is an optimal solution for us in terms of speed of data searching. A response time of our question answering system is between 2,5 and 5 seconds. The results are not very impressive but it is still enough that users don't need to wait for answers for a long time.

IV. Using Question Templates

Natural language processing is a domain of Computer Linguistics [10]. Programs and algorithms should behave like they understand natural language [5]. Natural language is ambiguous and contains many synonyms, which can be understood differently, depending on the context of the sentence or even paragraph. The key to understand the importance of the sentence is identification of entities. Methods for determining the meanings of phrases are generally based on the use of a large knowledge corpus. Most of those methods are slow, since they use a large amount of data and the results are average. This applies to the Slovenian language since there is currently no good enough semantic dictionary for it [6]. Therefore, we used a completely different approach and introduced the question templates in the context of domain-specific knowledge.

Question templates are a bridge between sentences and ontology. They are used for approximative substitution of

relations between entities in our ontologies. Templates can be equated with the ontology as formal presentation skills in the context of a domain. Elements of the question templates are entities composed of phrases, synonyms, class properties or even method parameters. We have basic and complex templates. Basic templates are composed of a question template that is related to a single answer template.

Example of basic question template:

What is the e-mail address of ***[Person_FirstName]*** ***[Person_LastName]****?*

What | *is* | *the* | *of* – words
e-mail address – phrase
[Person_FirstName] – ontology driven data that represents a class Person and its property Name
[Person_LastName] – ontology driven data that represents a class Person and its property Surname

Question template above has only one answer template:

[Person_FirstName] ***[Person_LastName]*** *e-mail address is* ***[Person_Email]****.*

e-mail address – phrase
[Person_FirstName] – ontology driven data that represents a class Person and its property Name
[Person_LastName] – ontology driven data that represents a class Person and its property Surname
[Person_Email] – ontology driven data that represents a class Person and its property Email

In complex cases, when the user didn't provide enough information for a unique response, we have to ask a supplementary question. This way the question templates can be related to the template of the second question (sub-questions). This approach leads us to our question answering dialog representation.

Example of complex question template (question answering dialog):

What is the price of a ***[Phone_Manufacturer]*** ***[Phone_ModelName]*** *when using* ***[Phone_PackageName]*** *package?*

In this example we won't be explaining question template entities, since we have done that in the previous example. When the user doesn't provide all three needed information (ontology driven data), the question template above cannot return a simple answer template. Because of this we have to build supplementary sub-questions carefully. In our case these sub-questions should be:

a) What is the manufacturer name of your cell phone?

b) What is the model name of your cell phone?

c) What is the package name you are using?

When the user enters all information we needed, then we can start exploring the returned answer template. In our case an answer template should look like this:

The price for a ***[Phone_Manufacturer]*** ***[Phone_ModelName]*** *when using* ***[Phone_PackageName]*** *package is* ***[Phone_ReturnPhonePrice]*** *EUR.*

We can see that this complex template also includes the use of an external resource. ***[Phone_ReturnPhonePrice]*** represents a call to a web service method named *ReturnPhonePrice*. Unfortunately, from this template, it cannot be seen what parameters are used by the method. We will dig into this in the next chapter.

V. External Knowledge Resources

As our system was developed as an applicative project, we had a special request. All information that our question answering system can handle, cannot be presented as an ontology. Certain information must nevertheless be calculated and that means we have to obtain that certain data from an external source.

The most logical approach was to use web services. Web services are typically application programming interfaces or Web APIs that are accessed via Hypertext Transfer Protocol (HTTP) and executed on a remote system hosting the requested services.

Because we couldn't get all the information as web services, we had to extend that process to other ways of calling methods. At the end we added the ability to call local DLL libraries and stored procedures from the database. As all three ways require method calls with the parameters, we need to provide a way to describe the methods and parameters. We consider it ideally to describe them with semantics. So we expanded our ontology representation with method and parameter description.

As we know from previous chapters, we have mapped our ontology to a relational database. These three external resources are totally different in a way of calling methods. Because we described all three methods with one database table, we had to find a perfect way to describe them. At the end we decided to describe them with XML technology. All of those three methods are unique and require different element description:

- *Web service* – web service URL address, service name, method name, parameters.
- *DLL library* – DLL binary data, class name, method name, parameters.
- *Stored procedure* – connection string, command name,

parameters.

A special attribute in a database table is designed to record this XML described information. While we designed very flexible semantic description of methods and parameters, our system is ready for expansion to new external resources.

Parameters in XML descriptions are only pointers to the real description of the parameter. A parameter is stored in special database table and is linked to the class property. We represent method parameters as a special class with properties. We already mentioned that a property is extended with regular expression. When we are using an external method with parameters of different data type, it is very convenient to verify data with regular expressions before we even call the method.

Because we now had three dissimilar ways of calling the methods, we also had to develop a special software wrapper. The software wrapper must figure out, from the semantic description, how to call methods with certain values for parameters and how to return and transform the returned calculated values.

VI. User Tracking

In applications that run as web applications we don't have any control about user inputs, so it is wise to track all user activities during the use of application.

Our system offers active and passive user tracking. Active tracking allows users to tell their opinions about a certain answer with a simple button click. The system even allows users to comment replied answers. Meanwhile, the passive tracking has a whole different approach which is very unobtrusive and users don't even know about it. Passive tracking uses client-side cookies for anonymous user tracking. We are trying to detect a context switch, which tells us about users' satisfaction.

With such actions we can constantly update our knowledge database. We can even track user questions and answers returned by our system. If we encounter a certain amount of errors in the responses, we can take appropriate action such as template rebuilding or restructuring of semantic data representation.

On the other hand we can generate all kinds of statistics that help us understand our users' behavior.

VII. Question Answering Process

The question answering process always starts with the user's input. The whole question answering process is shown in Fig. 3. When the user enters a sentence, the process for detecting entity candidates is executed. We already know that templates are composed of entities and that's why we have to find the appropriate candidates for the template matching process. An entity candidate detection process uses our domain specific knowledge database for detecting entities. Entities are recorded as instances of our ontology. At that stage our process uses a dialog states table, where the actual state of user dialog is stored. That is very important for understanding what data has already been entered by the user. We can detect individual entities (words, word phrases, ontology-driven data) using lemmatization [9]. This process is time-consuming and extends the entire question answering process. Instead of lemmatization process we introduced a whole new and different approach. The new algorithm is based on sequence matching with subsequence analysis [8]. We all are aware that user inputs can sometimes be incomprehensible because of typing mistakes or using a dialect in sentences. In these situations the lemmatization process often fails, while the sequence matching with subsequence analysis process gives more accurate results.

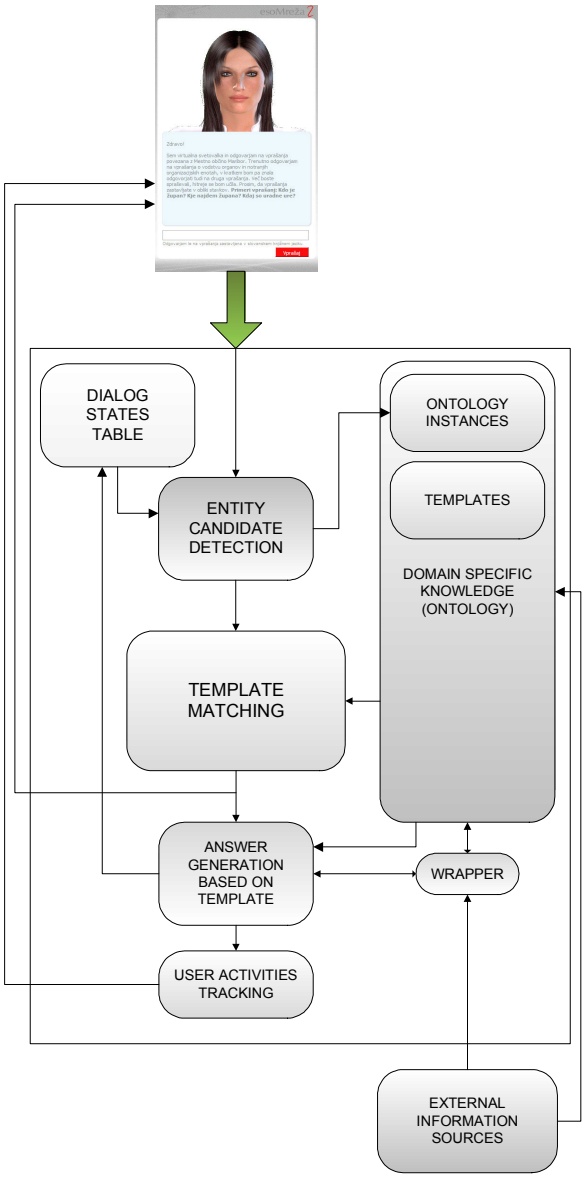

Fig. 3 Question answering process

The sequence matching with subsequence analysis algorithm operates in comparing subsequences with entities which are stored as ontology instances in the ontology database. The

algorithm is very complying and allows all sorts of typing errors as long as the percentage of success is greater than or equal to 70% of matching entities. This limit was defined by running algorithm through a large corpus of misspelled words.

Because we are still in a development phase, we can raise this percentage anytime we like. The entity candidate detection process only finds understandable words that are similar to the ontology instances. The sequence matching with subsequence analysis algorithm is not designed to work only with the Slovenian language, but it also works with other languages (i.e. English language).

The most important task in the whole process is the template matching process. This process must decide which question template is the most similar to the sentence entered by the user. Entity candidates and a dialog state list help that process to find the best calculated question template. A simplified Lesk algorithm is used to determine the correct meaning of each word in a given context by locating the sense that overlaps the most between its ontology definition and the given context [14]. Rather than simultaneously determining the meanings of all words in a given context, this approach tackles each word individually, regardless of the meaning of the other words occurring in the same context. The algorithm compares all entity candidates with entities from each question templates. We have to consider that some entities can be driven directly from ontology instances and others from set of entities returned from external resources. The matching entities are summarized. The highest percentage of summarized matching provides a resultant question template.

Question templates are also ranked, which helps us restrict our choice. In case the algorithm matching result is less than 80% system treated this as no match found. This situation does not lead to a one-way street, but gives the user advice on which question template to use.

Now, when the appropriate question template is found, we can generate an answer related either to an answer or a sub-question template.

If the entity in the template is represented as an external information resource, we have to a find semantic description of that source in our ontology. An external information resources can be a web service, a DLL library or a stored procedure in a database. A specifically written software wrapper then calls the appropriate method, which returns the result that represents the required entity values.

Before the answer is shown to the user, a special process records the user activity and alters the dialog state table. If the dialog with the user is not finished yet, an answer is formulated as a question. At that stage we have entered the question answer dialog.

VIII. User Interface

In the design field of human-computer interaction, the user interface is a place where interaction between humans and machines occurs. The goal of interaction between a human and a machine at the user interface is an effective operation and control of the machine, and feedback from the machine which aids the operator in making operational decisions.

Our question answering system is built as an application that needed two different parts of user interface for two totally different types of users. The most important part is designed for the general public users. These users can only input sentences and wait for an answer responded by the server. We are aware that users are not accustomed to such systems, but rarely using the search engines. Somehow we need to make sure they start communicating with the system as they would be talking to another person.

For this purpose we designed a flash animated female character so that users would imagine talking to a virtual person (fig. 4). The animated character can simulate facial expressions responsive to the user's input. The graphical user interface is also equipped with an input text box and a large panel for displaying returned answers. At the end of each answer response users can tell their opinions about the answer with a simple button click.

User interface clearly shows that question answering system indicates desire for using voice user interface.

Fig. 4 Main user interface

The second part of user interface is adjusted for ontology generation (fig. 5). Many would think this is not an important part of application, but they are wrong. Despite this part of user interface is named the administration, it is much more than that. The ontology generation process is very demanding. This process needs a lot of expertise and for domain specific knowledge there is a very small amount of experts. Therefore the user interface should be accommodated to them.

The user interface in figure 5 looks very busy. The fact is that we were not able to provide the optimal user interface for generation of ontologies. Whereas we initially used Protégé, there are visible similar guidelines in the user interface. At the beginning we tried to design a totally different and easier user interface, but it is hard to beat the many times used and tested Protégé user interface.

While everything in the user interface is attached to ontology, we designed a special control for that purpose. Because ontologies are hierarchically organized, we developed a special TreeView control. This control is capable of showing the whole hierarchical structure of our ontology including: domains, processes, classes, properties and methods. The TreeView control is quickly filled with hierarchical data and the whole control become useless in term of navigating through ontology classes. Therefore we upgraded the TreeView control with a smart filtering engine.

The right side of user interface is designed for input forms. But everything is not as simple as it looks. We had to design a Tab control which can handle multiple forms on various tabs. When the user selects an element in the TreeView control, certain tabs become available or unavailable depending on the selected element.

Inserting ontology instances is a slow and time-consuming process, and nobody likes it. It's fairly easy to create classes and properties, but the insertion of thousands of instances is really hard work. Because of this we added the ability for importing instances from comma separated text files and MS Office Excel files.

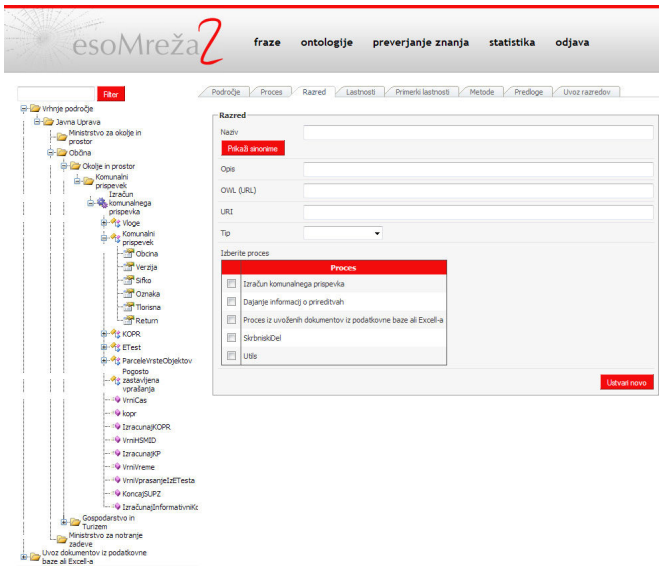

Fig. 5 Ontology generation user interface

At the time of writing this article, the user interface is available only in Slovenian language.

IX. Conclusion

This article describes our ontology-driven question answering system with semantic web services support. While we didn't want to build large knowledge corpuses of Slovenian language, we decided to semantically describe our domain-specific knowledge. The key component to our system is a well defined and semantically described ontology based knowledge database. Although there are some methods for storing ontologies, we built our own ontology mapping to relational database.

Because the question answering system should somehow understand natural language we managed to provide question templates. Question templates are a bridge between sentences and ontology. The template matching process is the most important part in our system. This process is responsible for the entire conversation dialog. The answer generation process is also built on top of the question templates. Some entities in those templates should be filled from ontology instances or even from external knowledge resources like web services. Our question answering system also tracks user's behavior.

A challenge for our future work is to improve algorithms for entity candidate detection and to speed up the algorithm for finding the minimum distance in question templates.

In the same question template ontology-driven data can represent only properties of the same class. We found this is not convenient in every situation. For that purpose we will have to append our ontology representation with relations between classes and properties. This will allow us using mixed class properties in the question templates. We are also considering introducing a specially defined language for the question templates. This language can drastically change the algorithm for finding the minimum distance in question templates.

A special treatment will be given to expand the set of external resources. We have built a scalable system that allows expansion of external resources. The priority on that will be given to Java class methods.

The introduction of relations between objects will most likely change the user interface. Therefore user interface will share some redesign. We also consider translating user interface into the English language.

Our ontology based knowledge database should always grow. You can get good results only if you have a large enough and quality knowledge corpus.

References

[1] G. Antoniou, F. Harmelen, *Web Ontology Language: OWL*

[2] I. Čeh, M. Ojsteršek, *Developing Question Answering System for the Slovene Language*, WSEAS transactions on information science and applications, Issue 9, Volume 6, September 2009.

[3] *The Protégé Ontology Editor and Knowledge Acquisition System*, http://protege.stanford.edu, visited on October 2010.

[4] N. Shadbolt, W. Hall and T. Berners-Lee, *The Semantic Web Revisited*, IEEE Intelligent Systems, Volume 21, No. 3, 2006.

[5] E. Sneiders, *Automated Question Answering Using Question Templates That Cover the Conceptual Model of the Database*, Proceedings of the 6th International Conference on Applications of Natural Language to Information Systems-Revised Papers, 2002.

[6] I. Čeh, M. Ojsteršek, *Slovene Language Question Answering System*, Proceedings of the 13th WSEAS International Conference on COMPUTERS.

[7] T. M. T. Sembok, H. B. Zaman, R. A. Kadir, *IRQAS: Information Retrieval and Question Answering System Based on A Unified Logical-Linguistic Model*, 7th WSEAS Int. Conf. on ARTIFICIAL INTELLIGENCE, KNOWLEDGE ENGINEERING and DATA BASES (AIKED'08).

[8] M. Ferme, M. Ojsteršek, *Sequence matching with subsequence analysis*, Proceedings of the European Conference on Advances in Communications, Computers, Systems, Circuits and Devices (ECCS'10).

[9] J. Brezovnik, M. Ojsteršek, *TextProc – A Natural Language Processing Framework*, Proceedings of the European Conference on Advances in

Communications, Computers, Systems, Circuits and Devices (ECCS'10).
[10] Y. Ledeneva, G. Sidorov, *Recent Advances in Computatonal Linguistics*, Informatica, Issue 34, 2010.
[11] D. Lavbič, M. Krisper, *Facilitating Ontology Development with Continuous Evaluation, INFORMATICA*, Volume 21, No. 4, 533-552, 2010.
[12] L. W. Lacy, *OWL: Representing Information Using the Web Ontology Language*, Trafford Publishing, 2005.
[13] J. G. Breslin, A. Passant, S. Decker, *The Social Semantic Web*, Springer, 2009.
[14] F. Vasilescu, P. Langlais, G. Lapalme, *Evaluating Variants of the Lesk Approach for Disambiguating Words*, LREC, 2004.

B. Gorenjak is a teaching assistant at University of Maribor, Faculty of Electrical Engineering and Computer Science. He graduated in 2000 and received his Master's degree in 2004 at Faculty of Electrical Engineering and Computer Science at University of Maribor. His research interests are natural language processing, Computer Human Interaction, Question-answering systems, ontologies and semantic web. He has been involved in several research and commercial projects on question-answering systems.

M. Ferme is a teaching assistant at University of Maribor, Faculty of Electrical Engineering and Computer Science. He graduated in 2008 at Faculty of Electrical Engineering and Computer Science at University of Maribor. His research interests are natural language processing, Question-answering systems, ontologies and semantic web. He has been involved in several research and commercial projects on question-answering systems.

M. Ojsteršek is an associate professor at University of Maribor, Faculty of Electrical Engineering and Computer Science. His research is focused on heterogeneous computing systems, semantic web, service-oriented architecture, natural language processing and dialog systems.