

From Semantic Web Knowledge To A Functional Conversational Agent: A Practical Approach

Tudor-Alexandru Dobrila
td108@ic.ac.uk

Abstract. This paper describes a practical approach for enhancing a language independent conversational agent for question answering using Semantic Web knowledge. Our main contribution is the scalable, cascade-type architecture, divided into several components. The first is a chat bot built on top of AliceBot, capable of providing fast and accurate responses in most cases. This is done by converting the ontology to AIML (Artificial Intelligence Markup Language [2]) format. The second is the ability to build answers from the ontology, by parsing and categorizing the user's input. This part works as a backup for the first component and significantly improves the accuracy of the answers.

Key words: artificial intelligence, chat bot, semantic web, ontology, AliceBot, machine learning, AIML, conversational agent

1 Introduction

The framework we propose in this paper is motivated by the task of building a smart, language independent, scalable conversational agent, capable of answering questions. Progress concerning this aspect of artificial intelligence has been made in the last years[1, 3]. Toward this end we have constructed a fully functional chat bot, which should provide quick and accurate results to the user's input.

Our key contribution is the introduction of a *cascade of agents*, where each one of them acts as a backup for the previous n agents. The best performing in terms of speed and accuracy of the answer are placed at the top of the chain. We categorized the questions into *domains* (as defined by Sumbaly et al. [3]), in order to treat as many input types as possible.

The first agent relies on converting Semantic Web knowledge to AIML [2] format, which is motivated by the work of E. Freese [1]. Our system is capable of obtaining approx. 15.000 AIML categories in 2 minutes from an ontology file (OWL) containing about 1000 entities. The main advantage of this approach is the speed of answer and flexibility, which is provided by the use of wildcards ("*") and underscores ("_"). This way 15.000 patterns actually cover millions of questions.

The second agent tries to detect the domain of the input and build an answer by querying the ontology. A lexical database is also used to analyze a wider range of inputs, by searching for synonyms.

We also present a preprocessing phase, where we expand the ontology using data from a lexical database and Wikipedia, in order to enrich the answers.

The remainder on this paper describes experimental results in several domains, such as mathematics, artificial intelligence and quantum mechanics.

2 General Architecture

We introduce in this paper the notion of *cascade of agents*. Instead of simply developing one chat bot, a chain of conversational agents can be created, where each one of them acts as a backup for the previous n agents. The best performing in terms of speed and accuracy of the answer, as tested empirically, are placed at the top of the chain. This assures us that if one component fails to deliver a response, another one will be considered, thus greatly improving the chances of obtaining a valid answer.

A general overview of the architecture can be seen in Fig. 1 .

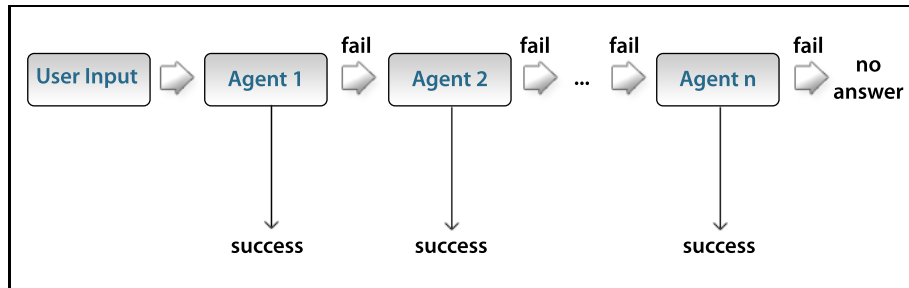


Fig. 1. A cascade of conversational agents.

In our approach, we created two conversational agents, one built on top of AliceBot and another one that interrogates the ontology itself, but several other agents can be added to this cascade. We tried to treat as many question domains as possible in both chat bots.

Protege API ¹, along with a reasoning engine capable of inferring logical consequences from a set of axioms (e.g. Pellet ²), is one of the most efficient ways to query the ontology.

2.1 Question domains

Based on the concept of *domains* (as defined by Sumbaly et al. [3]), we identified the following types of questions:

- Definition questions (e.g. 'What is X?')

¹ <http://protege.stanford.edu/plugins/owl/api/>

² <http://clarkparsia.com/pellet>

- Measure questions (e.g. 'How many types of X do you know?')
- List questions (e.g. 'What types of X do you know?')
- Comparison questions (e.g. 'What is the difference between X and Y?')
- Factual questions (e.g. 'Which X perform some Y operation?')
- Reasoning questions (e.g. 'Why does X ...?')

3 Agent 1: Converting the Ontology to AIML Format

At the top of the cascade we placed an AliceBot chat robot, because it is the fastest and covers a wide range of inputs. Several open source implementations are available, out of which we chose one developed in Java. A general overview of how the conversion works is presented in Fig. 2 .

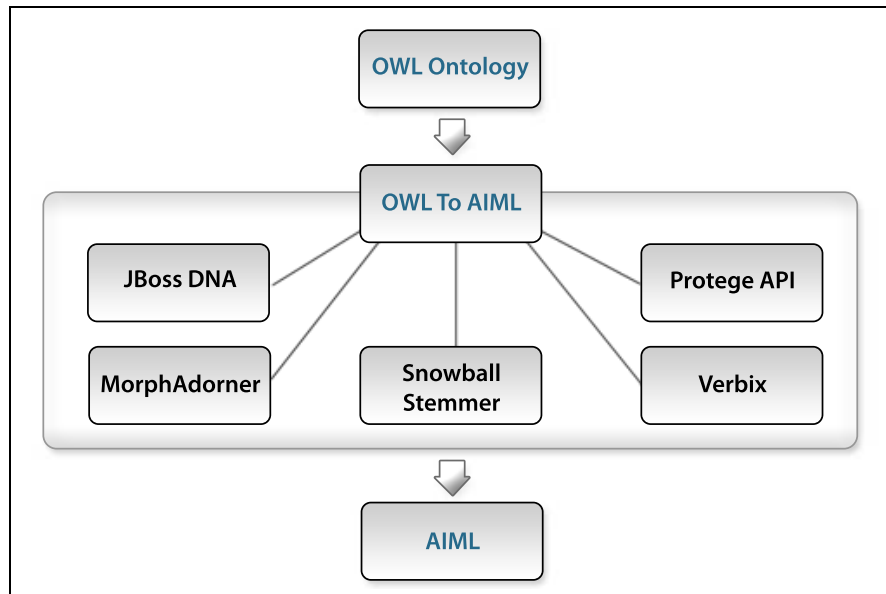


Fig. 2. The process of converting an ontology to AIML format.

3.1 Using Language Files With Generic Patterns

In order to make the system language independent and scalable, one XML file per language with generic patterns for each question domain is created. It contains template tags that are enclosed in squared brackets, which are replaced at runtime by actual values extracted from the ontology. They can also contain wildcards ("*") and underscores ("_"), which will be used by AliceBot.

For instance, for definition-type inputs in English, the XML would have the following form:

```

<patterns>
  <definition>
    <pattern>_ [name]</pattern>
    <pattern>What is * [name]</pattern>
    <pattern>Who is * [name]</pattern>
    <answer>[name] is a type of [value]</answer>
  </definition>
  ...
</patterns>

```

The syntax of the XML above is self-explanatory.

3.2 Generating AIML Patterns From Generic Patterns

The patterns are generated by querying the ontology graph with Protege API and then replacing the template tags defined in the language files.

- Definition questions (e.g. "What is * *X*?"): the superclasses of the *X* class and its restrictions are retrieved from the ontology and are used to create an answer.
For instance, for a class named **queue**, which belongs to **data structure** and **collection** and has a restriction **keeps_elements_in some order**, a conversation could be of the form:
Q: "What is a queue?"
A: "A queue is a type of data structure and collection. It keeps elements in order."
- Measure questions (e.g. "How many types * *X* *?"): the list of subclasses of *X* is retrieved from the ontology and is used to build an answer.
For instance, for a class named **variable_order**, a conversation could be of the form:
Q: "How many types of variable orders do you know?"
A: "I know two types of variable orders : dynamic variable order and static variable order."
- List questions (e.g. "What types of *X* *?"): the process is the same as for measure-type questions.
- Comparison questions (e.g. "_ *X* * *Y*?"): the superclasses and restrictions of both *X* and *Y* are retrieved and are used to search for similarities / differences between the two classes. For instance, for two classes **depth_first_search** and **breadth_first_search**, a conversation could be of the form:
Q: "What do you know about depth first search and breadth first search?"
A: "I know the following about depth first search and breadth first search : they are both algorithms. The difference is that depth first search use stacks, whereas breadth first search use queues."

- Factual questions ("Which *X* fulfill the *Y* property?"): the subclasses and restrictions of *X* are retrieved. For each restriction *Y*, an AIML category is generated.
For instance, for a class **problem**, which has a subclass **constraint_satisfaction_problem** with a restriction **is_solved_by some inference**, a conversation could be of the form:
Q: "What types of problems are solved by inferences?"
A: "Types of problems that are solved by inferences : constraint satisfaction problems."

3.3 Creating Grammatically Correct Patterns

Ensuring that the exported patterns are grammatically correct is vital. Several operations need to be taken in consideration when generating categories:

- Word pluralization: For instance, if we have a class named **algorithm**, a valid question would be "What types of **questions** *?". A good tool for finding the plural of a given word is JBoss DNA, implemented in Java.
- Verb conjugation: Needed for converting object properties. For instance, if we have a class **problem** and a restriction of the form **is_solved_by some algorithm**, a valid question would be "Which problems **are** solved by algorithms?". This can be done in two steps:
 - First, the root form (lemma) of a verb needs to be computed. MorphAdorner, implemented in Java, can be used to perform this step.
 - Next, the lemma needs to be conjugated. A good on-line conjugation resource is Verbix, which can be accessed from within code by sending HTTP requests and parsing the result. It also has the ability to conjugate verbs in several languages.
- Finding the gender of a given word: In some languages (e.g. Romanian), the gender can affect other words in a sentence, so finding it is important. This can be achieved by consulting a lexical database for the processed language (e.g. RoWordNet for Romanian).

3.4 Exporting the Resulting AIML

The conversion implements the *decorator pattern*. The basic idea is to "decorate" the final AIML file with various question domains. This ensures the scalability of the system, making the task of integrating a new type of input trivial.

```
AIMLGenerator generator =
    new DefinitionQuestion(
        new CombinationQuestion(
            new ListQuestion(...),
            ...),
        ...);
generator.export("path_to_file.aiml");
```

The AIML file needs to be loaded into an open-source implementation of AliceBot.

4 Agent 2: Querying the Ontology

The process of finding an answer by querying the ontology is done in three steps, as shown in Fig. 3. We will not present it in detail, since it is somewhat similar to the one described in section 3. The main advantage of this approach is that a wider range of possibilities is covered, because synonyms of words in the input are also considered. This comes at a cost, increasing the total run time. First, the question needs to be parsed. To accomplish this, we maintain a list of generic patterns similar to the ones defined in section 3.1 (e.g. a list-type question could be of the form *What types of [focus] [parameters]?*). We find the question's domain, the focus and the list of parameters by matching the input against a generic pattern and then extract relevant information.

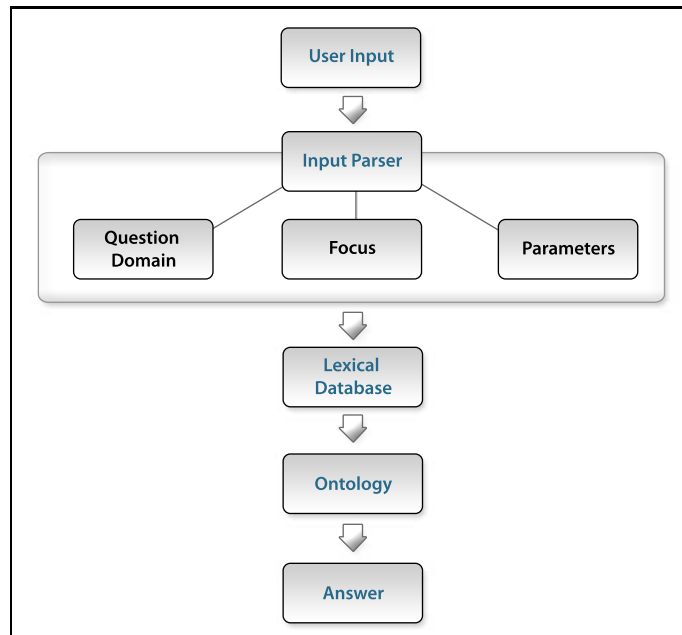


Fig. 3. The process of finding an answer from the ontology.

For instance, in *What types of problems are solved by inferences or by deductions?*, we can deduct that this is a list-type question (since it matches the above pattern), that the focus is the word *problems* and that the list of parameters is *{are solved by inferences OR are solved by deductions}*. We then expand the focus and list of parameters with synonyms from a lexical database.

After the question is parsed, the ontology is queried in order to find matches that could help answer the question. If both the focus and list of parameters have a match in the ontology, we can then proceed to building the answer of

the question. To ensure that it is grammatically correct, we apply the same principles as in section 3.3.

5 Ontology Expansion

Before any of the chat bots use the ontology, we applied a preprocessing phase. The entities in the OWL file are first annotated with definitions from a lexical database (in our tests we used Princeton's WordNet for English. For Romanian, RoWordNet could be used). This can be done in an unsupervised manner.

Next, relevant information from Wikipedia API ³ is extracted, by sending HTTP requests and parsing the content. This part is achieved in three steps and is done in a supervised manner:

1. Searching Wikipedia for a keyword
[http://en.wikipedia.org/w/api.php?action=opensearch&search=**keyword**](http://en.wikipedia.org/w/api.php?action=opensearch&search=keyword)
2. Extracting wikitext from pages related to that keyword
[http://en.wikipedia.org/w/api.php?action=query&prop=revisions&format=xml
&titles=**title**&rvprop=content&rvsection=T-0&redirects](http://en.wikipedia.org/w/api.php?action=query&prop=revisions&format=xml&titles=title&rvprop=content&rvsection=T-0&redirects)
3. Parsing wikitext and extracting only relevant information
[http://en.wikipedia.org/w/api.php?action=parse&format=xml&redirects
&text=**content**](http://en.wikipedia.org/w/api.php?action=parse&format=xml&redirects&text=content)

References

1. Eric Freese.: Enhancing AIML Bots Using Semantic Web Technologies. In Proc. of Extreme Markup Languages, 2007.
2. Richard Wallace.: Artificial Intelligence Markup Language (AIML). Working draft, A.L.I.C.E. AI Foundation, 2005.
3. R. Sumbaly, A. Kumar, G. Paruthi, S. Malhotra.: Artificially Intelligent Grid Assistant (A.I.G.A), International Conference on High Performance Computing, 2007.

³ <http://en.wikipedia.org/w/api.php>