Querying Cultural Heritage Knowledge Bases in Natural Language: Discussion Paper

Bernardo Cuteri, Kristian Reale, and Francesco Ricca

Department of Mathematics and Computer Science
University of Calabria, Italy
{cuteri,reale,ricca}@mat.unical.it

Abstract. Knowledge Based Question Answering (KBQA) is concerned with the possibility of querying data by posing questions in Natural Language (NL), that is by far, the most (human) common form of expressing an information need. This work reviews an approach to Question Answering that has the goal to transform Natural Language questions into SPARQL queries over cultural heritage knowledge bases. The key idea is to apply a rule-based classification process that we call template matching and that we have implemented in a prototype using logic programming. In the paper we discuss about the application of the prototype in real use-case, and indicate ongoing and future directions of work.

Keywords: Knowledge-based Question Answering · Cultural Heritage.

1 Introduction

Querying data from a database generally requires, at the very least, some programming expertise in the query language of the target database and a good understanding of the database model. Knowledge Based Question Answering (KBQA) is concerned with the possibility of querying data by posing questions in Natural Language (NL), that can be seen as the most (human) common form of expressing an information need.

In this work, we present an ongoing effort in the development of a KBQA system tailored to the cultural heritage domain, whose preliminary version has been developed within the PIUCULTURA project. The PIUCULTURA project, funded by the Italian Ministry for Economic Development, has the goal of devising a multi-paradigm platform that facilitates the fruition of cultural heritage sites in Italy. The prototype QA system described in this work is one of the core components of the platform, that answers to the need of providing a more natural way of obtaining information from the system.

It is worth noting that a system working on the domain of Cultural Heritage can benefit from many existing structured data sources that adhere to international standards. One of the most successful standards is the CIDOC Conceptual

Reference Model [9]. CIDOC-crm provides a common semantic framework for the mapping of cultural heritage information and has been already adopted as a base interchange format by museums, libraries, online data collections and archives all over the world [9,10]. For this reason, CIDOC-crm has been identified as the knowledge reference model for PIUCULTURA. Thus, our Question Answering prototype is applicable to query both general (e.g., online data collections) and specific (e.g., museums databases) CIDOC-compliant knowledge sources.

Our QA approach can be described as a waterfall-like process in which a user question is first processed from a syntactic point of view and then from a semantic point of view. Syntactic processing is based on the concept of *template*, where a template represents a category of syntactically homogeneous sentence patterns and is expressed in terms of Answer Set Programming (ASP) [12,7,6] rules. ASP is a well-established formalism for logic programming, and combines a comparatively high knowledge-modeling power with a robust solving technology [11,17]. ASP has roots in Datalog, that is a language for deductive databases.

On the other hand, the semantic processing is based on the concept of *intent*. By intent we mean the purpose (i.e., the intent) of the question: two questions can belong to two disjoint syntactic categories but have the same intent and vice versa. To give an example: *who created Guernica?* and *who is the author of Guernica?* have a quite different syntactic structure, but have the same intent, i.e., *know who made the work Guernica*. On the other hand, if we consider *who created Guernica?* and *who restored Guernica?* we can say that they are syntactically similar (or homogeneous), but semantically different: the purpose of the two questions is different. Semantic disambiguation, in which intents are mapped to a set of predefined queries on the knowledge base, is done by resorting to the multilingual BabelNet [20] dictionary.

2 Question Answering for Cultural Heritage

2.1 The problem

The goal of our approach is to answer questions on cultural heritage facts stored in a repository modeled according to a standard model of this domain. In particular, the target knowledge base model of the project is the CIDOC conceptual reference model [9]. The CIDOC-crm is a standardized maintained model that has been designed as a common language for the exchange and sharing of data on cultural heritage without loss of meaning, supporting the implementation of local data transformation algorithms towards this model. This standard has been adopted worldwide by a growing number of institutions and provides a more trustable, structured and complete source w.r.t. freely available (often unstructured and non-certified) web resources. Indeed, museums and institutions typically have structured sources in which they store information about their artifacts that can be mapped to CIDOC-crm rather easily (actually, this was one of the main goals of the promoters of CIDOC-crm). On the other hand, the availability of documentary sources is limited. If we take into consideration freely available documentary sources such as Wikipedia, we realize that the percentage coverage of works and authors is low. For example, a museum like the British Museum has about 8 million artifacts (made available in CIDOC-compliant format) while on Wikipedia there are in total about 500 thousand articles about works of art from all over the world. The CIDOC-crm is periodically released in RDFs format [8], thus our Question Answering system has to answer questions by finding the information required on an RDF knowledge base that follows the CIDOC-crm model. The reference query language of RDF is SPARQL [13]. So, in basic terms, the QA system has to map natural language questions into SPARQL queries and produce answers in natural language from query results.

2.2 Overview of the Approach

In this section we present step-by-step our question answering system for cultural heritage. In particular, the question answering process is split into the following phases: **(1) Question NL Processing**: the input question is transformed into a three-level syntactic representation; **(2) Template Matching**: the representation is categorized by a template system that is implemented by means of logical rules; **(3) Intent Determination**: the identified template is mapped to an intent, where the intent identifies precisely the intent (or purpose) of the question; **(4) Query Generation**: an intent becomes a query on the knowledge base; **(5) Query Execution**: the Query is executed on the knowledge base; **(6) Answer Generation**: the result produced by the query is transformed into a natural language answer. Splitting the QA process into distinct phases allowed us to implement a system by connecting loosely-coupled modules dedicated to each phase. In this way we also achieved better maintainability, and extensibility. In the following sections, we analyze in detail the 6 phases just listed.

Question NL Processing The NL processing phase builds a formal and therefore tractable representation of the input question. The question is decomposed and analyzed by highlighting both the atomic components that compose it, and the morphological properties of the components and the relationships that bind them. In this phase we perform the following NLP steps: Named entities recognition, Tokenization, Part-of-speech tagging, and Dependency Parsing.

Named entities recognition (NER) is an NLP task that deals with the identification and categorization of the entities that appear in a text. The named entities are portions of text that identify the names of people, organizations, places or other elements that are referenced by a proper name. For example, in the phrase *Michelangelo has painted the Last Judgment*, we can recognize two entities: *Michelangelo*, that could belong to a *Person* category, and *the Last Judgment* that could belong to an *Artwork* category. When the entities of the text have been recognized, they can be replaced with placeholders that are easier to manage in the subsequent stages of processing of natural language. In our implementation we use CRF++ [14] that implements a supervised model based on conditional random fields that are probabilistic models for segmenting and labeling sequence data [15].

Tokenization consists of splitting text into words (called tokens). A token is an indivisible unit of text. Tokens are separated by spaces or punctuation marks. In western languages, the tokenization phase turns out to be rather simple, as these languages place quite clear word demarcations. In fact, the approaches used for natural language tokenization are based on simple regular expressions. Tokenization is the first phase of lexical analysis and creates the input for the next Part-of-Speech Tagging phase.

The Part-of-Speech tagging phase assigns to each word the corresponding part of the speech. Common examples of parts of speech are adjectives, nouns, pronouns, verbs, or articles. The Part-of-Speech assignment is typically implemented with supervised statistical methods. There are, for several languages, large manually annotated corpora that can be used as training sets to train a statistical system. Among the best performing approaches are those based on Maximum Entropy. For tokenization and POS-tagging we used the Apache OpenNLP library[1] with pretrained models.

Finally, Dependency Parsing is the identification of lexical dependencies of the words of a text according to a grammar of dependencies. The dependency grammar (DG) is a class of syntactic theories that are all based on the dependency relationship (as opposed to the circumscription relation). Dependency is the notion that linguistic units, e.g., words, are connected to one another by directed connections (or dependencies). A dependency is determined by the relationship between a word (a head) and its dependencies. The methods for extracting grammar dependencies are typically supervised and use a reference tag-set and a standard input representation format known as the CoNLL standard developed and updated within the CoNLL scientific conference (Conference on Computational Natural Language Learning). In our implementation we used MaltParser[2] that is a system for data-driven dependency parsing [21].

Template Matching Template matching is in charge of classifying question from the syntactic point of view and extract the question terms that are needed to instantiate the query for retrieving the answer. Basically, a template represents a category of syntactically homogeneous questions. In our system, templates are encoded in terms of ASP rules. By using ASP we can work in a declarative fashion and avoid implementing the template matching procedure from scratch.

To this end, the output of the NLP phase is modeled by using ASP facts that constitute the input of the template matching module. In particular, words are indexed by their position in the sentence and they are associated with their morphological feature by using facts of the following forms:

```
word(pst,wrd).     pos(pst,pos_tag).     gr(pst1,pst2,rel_tag).
```

the first kind of fact associates position of words (`pst`) in a sentence to the word itself (`wrd`); the second associates words (`pst`) with their POS tags (`pos_tag`),

[1] `https://opennlp.apache.org`
[2] `http://www.maltparser.org/`

and the latter models grammatical relations (a.k.a. typed dependencies) specifying the type of grammatical relation (`rel_tag`) holding among pair of words (`pst1,pst2`). The possible tags and relations terms are constants representing the labels produced by the NLP tools mentioned in the previous subsection.

Consider, for example, the question *who painted Guernica?*, the output of the NLP phase would result in the following facts:

```
word(1, "who"). word(2, "painted"). word(3, "Guernica").
word(4, "?"). pos(1, pr). pos(2, vb). pos(3, np). pos(4, f).
gr(2, 1, nsubj). gr(2, 3, dobj). gr(2, 4, punct).
```

In the template matching phase, questions are matched against question templates. Templates identify categories of questions that are uniform from the syntactic point of view and we express them in the form of ASP rules.

Basically, each template rule models a condition under which we identified a possible syntactic question pattern for a template. An example of template rule that matches questions of the form *who action object?* is the following:

```
template(who_action_object, terms_2(V, O), 8) :-
word (1, "who"), word(2, V), word(3, O), word(4, "?"),
pos(1, pr), pos(2, vb), pos(3, np), pos(4, f),
gr(2, 1, nsubj), gr(2, 3, dobj), gr(2, 4, punct).
```

In the example, *who_action_object* is a constant that identifies the template, while $terms(V, W)$ is a function symbol that allows extracting the terms of the input question, respectively the verb V and the object O. The intuitive meaning of a template rule is that whenever all atoms in the body (the part of the rule affter `:-`) are true, then the template matches. The number 8, is the weight of the template rule. The weight is a number expressing the importance of a pattern. By using weights one can freely express preferences; for instance in our implementation we set this number to the size of elements in the body of the ASP template rule to favor more specific templates over more generic ones.

Question templates are intended to be defined by the application designer, which is a reasonable choice in applications like the one we considered, where the number of templates to produce is limited. Nonetheless, to assist the definition of templates we developed a graphical user interface. Such interface helps the user at building template rules by working and generalizing examples, and does not require any previous knowledge of ASP or specific knowledge of NLP tools. The template editing interface is not described in this paper for space reasons.

In our prototype, we used DLV [16] as engine that computes the answer sets (thus the best matches) of the template matching phase, and the DLV Wrapper library to [22] to manage DLV invocations from Java code. DLV has a Datalog subsystem supporting efficient ontological reasoning [4,2,5,3,1].

Intent Determination The identification of a question by templates might be not sufficient to identify its intent or purpose. For example, *who painted Guernica?* and *who killed Caesar?* have similar syntactic structures and may fall into the same template, but they have two different purposes. The intent determination process is based on the lexical expansion of question terms extracted in the template matching phase and has the role of identifying what the question asks (i.e., its intent), starting from the result of the template matching phase. In other words, it disambiguates the intent of questions that fall into the same syntactic category. In the previous example, painting is *hyponym* (i.e., a specific instance) of creating and thus we can understand that the intent is to determine the creator of a work, while killing does not have such relationships and we should, therefore, instantiate a different intent. In the same way, *who painted Guernica?*, *who made Guernica?* are questions that can be correctly mapped with a single template and can be correctly recognized by the same intent thanks to the fact that all three verbs are hyponyms or synonyms of the verb *create*. Words semantic relations can be obtained by using dedicated dictionaries, like wordnet [19] or BabelNet [20]. In our system we implemented the intent determination module in Java and used the BabelNet API library for accessing word relations. In particular, intent determination is implemented as a series of Java conditional checks (possibly nested) on word relations. Such conditional checks are expressed as a question term Q, a word relation R and a target word T. The BabelNet service receives such triple and returns true/false depending on whether Q is in relation R w.r.t. T, R is either synonymy, hyponymy or hyperonymy.

The implementation of intent determination is done by the designer as template definition. Our system implements a set of intents that were identified during the analysis by a partner of the project.

Query Execution The intents identified in the previous phase are mapped one to one with template queries, called prepared statements in programming jargon. In the Query Execution phase, the query template corresponding to the identified event is filled with the slots with terms extracted from the template matching phase and executed over the knowledge base. The CIDOC-crm specification is, by definition, an RDF knowledge base [8], thus we implemented the queries corresponding to intents in the SPARQL language [13]. In our prototype, we store our data and run our queries using Apache Jena, as programmatic query API, and Virtuoso Open-Source Edition as knowledge base service.

Answer Generation Finally, the latest phase of a typical user interaction with the QA system is the so-called Answer Generation. In this phase, the results produced by the execution of the query on the knowledge base are transformed into a natural language answer that is finally presented to the user. To implement this phase, we have designed answer templates that are natural language patterns with parameterized slots that are filled according to the question intent and the terms extracted from the database.

2.3 Preliminary experiments

We conducted an experimental analysis to assess the performance of the system on a set of 60 template rules, which are able to handle basic question forms and types for the cultural heritage domain distributed in 20 different intents.We report, on a sample of 167 questions, an average template matching time of 34 milliseconds (ms) on an Intel i7-7700HQ CPU architecture. For what concerns the other phases of the QA system, the NL phase average execution time is of 30 ms and is at most 50 ms, the intent determination phase average execution time is of 50 ms and is at most 580 ms and the average query execution time is of 8 ms and is at most 32 ms. Overall the system presents good execution times, which are acceptable for a real-time QA system.

3 Discussion and future works

In this work, we presented an approach to the problem of Knowledge-based Question Answering in the Cultural Heritage domain. The presented solution transforms input questions into SPARQL queries that are executed on a knowledge base. The concrete application of the approach to the Cultural Heritage use-case shows that one of the main advantages of the approach is that it is suited for fast prototyping, as it allows the fast integration of new question forms and new question intents. This feature suits the need of closed domains that are characterized by a limited number of intents and question forms. On the other hand a drawback of the approach is that it relies on the manual work of a designer that provides question templates and the strategy for intent matching. This problem is partly solved by some other QA approaches, as the ones based on machine learning that requires less engineering efforts in the implementation of question understanding at the cost of providing a corpora of examples.

The prototype can be seen as a good starting point for many future works. One of the most natural observation that comes in mind is that the intent determination phase can be also encoded using logic programs, thus making also the specification of this step flexible and independent from a static decision made at compile time. Indeed, having the whole question understanding process in ASP makes it more homogeneous and easier to develop and maintain. A prerequisite of implementing intent determination in ASP is to make ASP code interact with the dictionary that provides words semantic relations. We think that this can be achieved either by porting dictionary data in ASP or by using ASP extensions with external computation. Another interesting thing to investigate is how the ASP-based implementation compares with other techniques for question answering, as the ones only based on machine learning (e.g., [23,18]); and develop an architecture where the two can be combined in a single implementation.

References

1. Adrian, W.T., Manna, M., Leone, N., Amendola, G., Adrian, M.: Entity set expansion from the web via ASP. In: ICLP (Technical Communications). OASICS, vol. 58, pp. 1:1–1:5. Schloss Dagstuhl - Leibniz-Zentrum fuer Informatik (2017)
2. Amendola, G., Leone, N., Manna, M.: Finite model reasoning over existential rules. TPLP **17**(5-6), 726–743 (2017)

3. Amendola, G., Leone, N., Manna, M.: Querying finite or arbitrary models? no matter! existential rules may rely on both once again (discussion paper). In: SEBD. CEUR Workshop Proceedings, vol. 2037, p. 218. CEUR-WS.org (2017)
4. Amendola, G., Leone, N., Manna, M.: Finite controllability of conjunctive query answering with existential rules: Two steps forward. In: IJCAI. pp. 5189–5193. ijcai.org (2018)
5. Amendola, G., Leone, N., Manna, M., Veltri, P.: Enhancing existential rules by closed-world variables. In: IJCAI. pp. 1676–1682. ijcai.org (2018)
6. Bonatti, P.A., Calimeri, F., Leone, N., Ricca, F.: Answer set programming. In: 25 Years GULP. Lecture Notes in Computer Science, vol. 6125, pp. 159–182. Springer (2010)
7. Brewka, G., Eiter, T., Truszczyński, M.: Answer set programming at a glance. Communications of the ACM **54**(12), 92–103 (2011)
8. Consortium, W.W.W., et al.: Rdf 1.1 concepts and abstract syntax (2014)
9. Doerr, M.: The cidoc conceptual reference module: an ontological approach to semantic interoperability of metadata. AI magazine **24**(3), 75 (2003)
10. Doerr, M., Gradmann, S., Hennicke, S., Isaac, A., Meghini, C., Van de Sompel, H.: The europeana data model (edm). In: World Library and Information Congress: 76th IFLA general conference and assembly. pp. 10–15 (2010)
11. Gebser, M., Leone, N., Maratea, M., Perri, S., Ricca, F., Schaub, T.: Evaluation techniques and systems for answer set programming: a survey. In: IJCAI. pp. 5450–5456. ijcai.org (2018)
12. Gelfond, M., Lifschitz, V.: Classical negation in logic programs and disjunctive databases. New generation computing **9**(3-4), 365–385 (1991)
13. Harris, S., Seaborne, A., Prud'hommeaux, E.: Sparql 1.1 query language. W3C recommendation **21**(10) (2013)
14. Kudo, T.: Crf++. http://crfpp. sourceforge. net/ (2013)
15. Lafferty, J., McCallum, A., Pereira, F.C.: Conditional random fields: Probabilistic models for segmenting and labeling sequence data (2001)
16. Leone, N., Pfeifer, G., Faber, W., Eiter, T., Gottlob, G., Perri, S., Scarcello, F.: The dlv system for knowledge representation and reasoning. ACM Transactions on Computational Logic (TOCL) **7**(3), 499–562 (2006)
17. Lierler, Y., Maratea, M., Ricca, F.: Systems, engineering environments, and competitions. AI Magazine **37**(3), 45–52 (2016)
18. Lukovnikov, D., Fischer, A., Lehmann, J., Auer, S.: Neural network-based question answering over knowledge graphs on word and character level. In: Proceedings of the 26th international conference on World Wide Web. pp. 1211–1220. International World Wide Web Conferences Steering Committee (2017)
19. Miller, G.: WordNet: An electronic lexical database. MIT press (1998)
20. Navigli, R., Ponzetto, S.P.: Babelnet: The automatic construction, evaluation and application of a wide-coverage multilingual semantic network. Artificial Intelligence **193**, 217–250 (2012)
21. Nivre, J., Hall, J., Nilsson, J., Chanev, A., Eryigit, G., Kübler, S., Marinov, S., Marsi, E.: Maltparser: A language-independent system for data-driven dependency parsing. Natural Language Engineering **13**(2), 95–135 (2007)
22. Ricca, F.: A java wrapper for DLV. In: Answer Set Programming. CEUR Workshop Proceedings, vol. 78. CEUR-WS.org (2003)
23. Zhong, V., Xiong, C., Socher, R.: Seq2sql: Generating structured queries from natural language using reinforcement learning. arXiv preprint arXiv:1709.00103 (2017)