Automatic Question Pattern Generation for Ontology-based Question Answering

Shiyan Ou, Constantin Orasan, Dalila Mekhaldi, Laura Hasler

Research Group in Computational Linguistics
University of Wolverhampton, UK
{shiyan.ou, c.orasan, dalila.mekhaldi, l. hasler}@wlv.ac.uk

Abstract

This paper presents an automatic question pattern generation method for ontology-based question answering with the use of textual entailment. In this method, a set of question patterns, called *predictive questions*, which are predicted to be asked by users in a domain, were generated on the basis of a domain ontology. Their corresponding query templates, which can be used to extract answers to the predictive questions from a knowledge base, were generated as well. The process of producing these question patterns and query templates is described and discussed in the context of the "Movies & Cinemas" domain. An evaluation was carried out to assess the quality of the generated question patterns with the help of a textual entailment engine.

1. Introduction

The semantic web, which encodes some semantics of web-resources in a machine-readable form, is regarded as the future in the evolution of the World Wide Web. It offers an opportunity to develop novel, sophisticated forms of Question Answering (QA), where ontologies play a crucial role (Lopez et al. 2005). The common feature of such ontology-based QA systems is that they require the representation of both natural language user questions and information sources using formats compliant with a common ontology (Basili et al. 2004). Once unstructured information sources are marked up semantically and transformed into structured knowledge bases, well-structured queries, often written in a certain standard query language (e.g. SQL, SPARQL, and RQL), are often used to retrieve data from underlying knowledge bases. In order to draw correct answers from such sources, a natural language question needs to be precisely translated into such a query. However, this is a difficult task involving complex semantic annotation and knowledge representation. In addition to this, language is complex and ambiguous, and the same question can be asked using various expressions, for example, "*Where can I see movie X?*", "*Which cinema is showing movie X?*" and "*What is the name of the cinema which is showing movie X?*". These "different" questions share the same query for data retrieval and obtain the same answers, and thus it is not worth processing each expression to produce the same query. To address this problem, textual entailment was proposed as a solution to determine whether different expressions entail the same meaning and thus can use the same retrieval procedure (Kouylekov et al. 2006).

In our EU-funded project QALL-ME[1], which aims to establish a shared infrastructure for multilingual and multimodal QA in the tourism domain, we proposed an ontology-based QA method which uses a set of predefined question patterns for answering new questions with the use of textual entailment. In this project, a domain ontology was designed to provide a common vocabulary for the selected domain as well as a computerized specification of the meaning of terms used in the vocabulary. The unstructured tourism data taken from the web were semantically marked up using the ontology and converted into the triple-based RDF format. On the basis of the ontology, a set of question patterns, called *predictive questions*, which are predicted to be asked by users in the domain, were generated automatically, along with their corresponding query templates. Following Harabagiu and Hickl (2006), if a user question entails a predictive question, the answers of the predictive question are expected to be some subset of the answers to the user question. Thus for an input user question, we use textual entailment to discover the predictive question entailed by it, and then use the query template of the selected predictive question to produce a complete query for retrieving the answers to the user question. The main advantage of the method proposed is that we do not need to annotate the user questions which look different but entail the same meaning, and more precise queries can be created to retrieve correct answers without the need of deep question processing.

[1] http://qallme.fbk.eu

This paper focuses on the automatic generation of question patterns as well as their corresponding query templates on the basis of the domain ontology. An evaluation was carried out to assess the quality of the generated question patterns by using a textual entailment engine to find out the one entailed by a user question. The subsequent sections are organized as follows: Section 2 reviews some related question answering systems, Section 3 describes the domain ontology and the underlying ontological knowledge base, Section 4 describes the generation of predictive questions and their corresponding query templates, Section 5 reports the evaluation and its results, and Section 6 presents the discussion and conclusion.

2. Related Work

In question answering, the issue of representing a natural language question as a structured format for data retrieval is not a trivial task. Work in this area can be traced back to the early database-based QA systems. Using linguistic processing, Chat-80 (Warren and Pereira 1982) transforms a natural language question into a ProLog query, whereas RECISE (Popescu et al. 2003) maps a question to an SQL query. In the open-domain QA system START (Katz et al. 2002), a natural language question was translated into a database query in the form of <*object property value*>, where the value of the object's property represents the expected answer to the question. For example, the question "*Who directed Gone with the wind?"* is represented as <*'imdb.movie' 'Gone with wind (1939)' 'DIRECTOR'*>.

The availability of domain ontologies makes possible to semantically annotate questions and transform them into an ontological representation. An early ontology-based question analysis was investigated in the EU project MOSES (Atzeni et al. 2004). In this QA system, a natural language question is first represented as a Question Quasi-Logical (Q-QLF) form with syntactic analysis. Then, a domain ontology, which has been mapped to a general linguistic resource (e.g. EuroWordNet), is used to map the Q-QLF form into an ontology-based concept-relation form for data retrieval from the semantically structured web contents. Another ontology-driven QA system, Aqualog (Lopez et al. 2005), makes use of linguistic tools (e.g. GATE) and resources (e.g. WordNet) to annotate the terms and relations in a natural language question and then translates the question into a set of <*subject, predicate, object*> triples. These intermediate triples are further processed to produce ontology-compliant logical queries for drawing the answers from a knowledge base with ontology-compliant semantic markup. The third demonstration system which uses ontology-based full knowledge representation to answer questions is the KSL Wine Agent[2]. This web service poses a structured query expressed in OWL DL to the knowledge base which contains structured content information, and uses a reasoner to check whether the content matches the query. A representative question which can be answered by the system is "*What kind of wine should I serve with a meal whose main course is pasta with spicy red sauce?*"

A typical example of using predefined questions and textual entailment in QA is described in Harabagiu and Hickl (2006). First, the search engine of their QA system returned a set of ranked passages in response to a user question. To extract correct answers from these passages, a set of possible questions with the associated answers was generated automatically from the top-ranked passage. Using textual entailment, the predictive questions which were entailed by the user question were recognized and their answers were used as the correct answers to the user question. Our work is similar to that of Harabagiu and Hickl (2006). However, in contrast to their work, (1) we generated question patterns and the associated query templates rather than concrete questions and the associated answers; (2) we generated predefined questions based on a domain ontology instead of the passages which were expected to contain the answers; and (3) our work was used for restricted-domain QA whereas their work was for open-domain QA.

3. Ontology and Ontological Knowledge Base

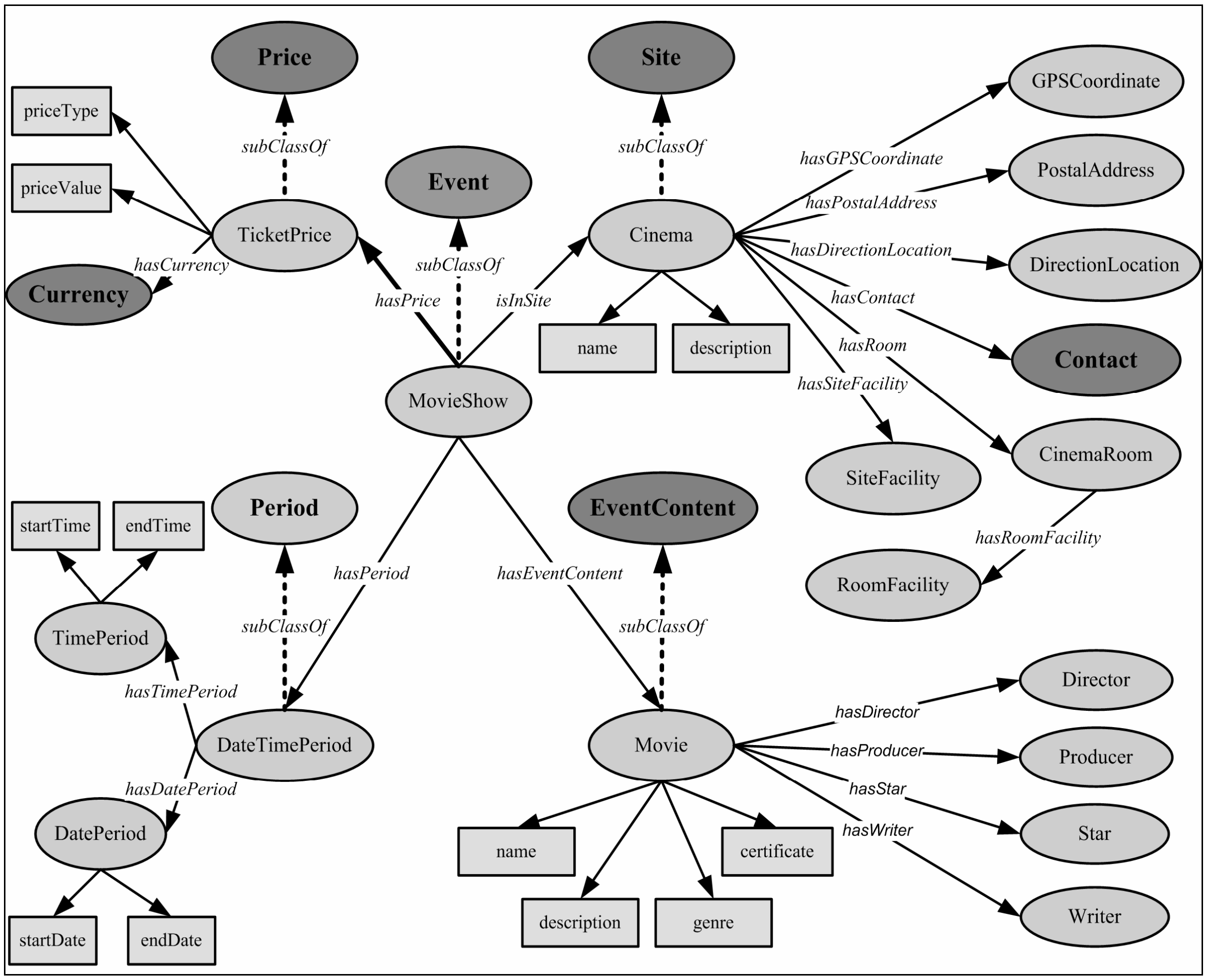
The ontology designed in our project aims at providing a conceptualized description of the tourism domain. It mainly covers tourism sites and tourism events in certain destinations (cities or towns). The ontology was encoded using the OWL DL language. A representative part of the ontology is the sub-domain of "Movies & Cinemas", shown in Figure 1. It involves a type of tourism site, *Cinema*, and a type of tourism event, *MovieShow*, and focuses on the relationships between them.

From the point of view of design, the top-level classes fall into three categories:

- **Main classes** refer to the most important concepts in the tourism domain, e.g. *Site*, *Event* and *EventContent*.
- **Element classes** refer to the elements of the main classes or the elements of other element classes, e.g. *Room* (including *CinemaRoom*, *GuestRoom* etc.), *Facility* (including *SiteFacility* and *RoomFacility*), *PersonOrganization* (including *Star*, *Director* etc.).
- **Attribute classes** refer to the packages of a group of attributes of the main classes or element classes, e.g. *Contact*, *Period*, *Price* and *Location* (including *GPSCoordinate*, *PostalAddress*, *DirectionLocation*).

From an application perspective, the instances of the main classes can exist independently whereas the instances of the element classes and attribute classes have to be attached to the instances of the main classes or other element classes.

[2] http://ksl.stanford.edu/people/dlm/webont/wineAgent/



- *The ellipse boxes represent classes and the rectangular represent literals; the top-level classes are highlighted in bold.*
- *The solid lines represent object properties and datatype properties, and the dotted lines represent subClassOf.*

Figure 1. A part of the tourism ontology on the sub-domain of "Movies & Cinemas"

To construct a knowledge base for providing potential answers to user questions, we annotated the original tourism data obtained from the web using semantic markup derived from the designed OWL ontology. The tourism data were encoded in the RDF/XML format, which can be used to instantiate the ontology. The RDF/XML documents were persistently stored in the MySQL relational database as an RDF data model in the triple form of <*subject, predicate, object*>. To access the RDF-based knowledge base, RDF query languages, such as SPARQL and RQL, need to be used to retrieve specific contents from it for QA. This procedure can be viewed as an extension of pattern matching.

4. Automatic Generation of Predictive Questions with Query Templates

In this section, we focus on how to generate natural language predictive questions which can be asked and answered based on a domain ontology. Although this paper focuses on the sub-domain of "Movies & Cinemas", the method can be generalized for the whole tourism domain. In addition, we use the SPARQL query language as the example to show how to generate corresponding query templates for retrieving answers to these predictive questions. Templates using other query languages can be generated in a similar way.

Like the RDF model, SPARQL is also built on triple patterns and is written in the form of a subject, predicate and object, but must be terminated with a full stop. In a SPARQL triple pattern, any of the subject, predicate, and object values may be replaced by a variable that is denoted using a question mark (e.g. *?directorName*). Variables are used to indicate data items of interest that will be requested by a question (for more details about SPARQL, please see http://www.w3.org/TR/rdf-sparql-query/)

In the OWL ontology, a class usually has a list of associated properties which can be applied to the instances of the class. The value range of the property is usually specified by the global range restriction <*rdfs: range*>, but it may be narrowed by the local range restriction <*owl: allValuesFrom*> with respect to a particular class. A class, an associated property, and the range of the property on the class can be written in the form of the RDF triple, i.e. a 3-tuple of subject, predicate and object, where the subject represents the class, the predicate represents the property, and the object represents the value range of the property, for example, <*MovieShow, isInSite, Cinema*>. If the range is a union class containing more than one named classes, each of these classes is regarded as an object for creating a triple, e.g. <*MovieShow*, *hasPeriod*, *DateTimePeriod*> + <*MovieShow*, *hasPeriod*, *DatePeriod*>.

Since the element and attribute classes need to be attached to the main classes, we started from the main classes to create predictive questions and query templates. For a main class, we used Jena[3], a Semantic Framework for Java, to parse the OWL ontology and derived all the properties associated with it to write each in the triple form: <*class, property, range*>. These associated properties represent all the possible items which can be queried for the instances in the class. Among these properties, we first located the one which is the most frequently asked by users. By analyzing 100 randomly selected user questions, we found that for most of the main classes (e.g. *Movie* and *Cinema*) the property is 'name'. Thus, for the instances in such main classes, we create two types of queries focusing on the 'name' property:

- **T1:** Query the 'name' property of a class instance using one or more of its other properties as the constraint(s). If only one property 'xxx' is taken, the one-constraint template is defined as follows:
  - What is the name of the <class> which has the <xxx> [<xxx>_value]?
- **T2:** Query a property 'xxx' (different to 'name') of a class instance using its 'name' property as the constraint. The T2 template is defined as follows:
  - What (or When, Where, How long) is the <xxx> of [<class>_name]?

In the above templates, the angle-brackets slots represent the names of the classes or properties which will be filled while producing predictive questions. The square-brackets slots represent the name of the entities (e.g. *movie genre*, *movie name*) defined in the domain ontology, which will be filled with the real values in a user question while producing its complete SPARQL query.

If the property 'xxx' is a datatype property (e.g. *genre*), its range is a literal (e.g. *string*). For each type of query with one constraint, one natural language predictive question is produced. At the same time, its query template is generated based on the two triples: <*class, name, string*> and <*class,* xxx*, range*>. For example, if class='Movie', xxx='genre', the following two predictive questions are generated based on the above templates, along with the corresponding SPARQL query templates.

[3] http://jena.sourceforge.net

T1-1: *What is the name of the movie which has the genre [genre_value]?*

```
SELECT   ?movieName
WHERE {
?Movie   prefix:name      ?movieName.
?Movie   prefix:genre     "[genre_value]"^^<xsd:string>. }
```

T2-1: *What is the genre of [Movie_name]?*

```
SELECT   ?genreValue
WHERE {
?Movie     prefix:name      "[Movie_name]"^^<xsd:string>.
?Movie     prefix:genre     ?genreValue.                 }
```

If the property 'xxx' is an object property (e.g. *hasPostalAddress*), its range is a class (e.g. *PostalAddress*). Then the triples associated with this class are created subsequently using Jena. The procedure is repeated until the objects in all the triples are literals. Each of the triples will be used to create a SPARQL triple pattern. For example,

```
<Cinema, name, string>
<Cinema, hasPostalAddress, PostalAddress> →
          <PostalAddress, street, string>
          <PostalAddress, postalCode, string>
          <PostalAddress, isInDestination, Destination> →
                              <Destination, name, string>
```

For the first type of queries, T1, three predictive questions can be produced, each of which uses one of the properties associated with the range class as the constraint.

T1-2: *What is the name of the cinema which is in [street_value]?*
T1-3: *What is the name of the cinema which has the postal code [postalCode_value]?*
T1-4: *What is the name of the cinema which is in [Destination_name]?*

For the second type of queries, T2, four predictive questions can be produced as follows. Each of the first three questions queries one of the properties associated with the range class, and the last one queries the overall range class (i.e. all of its associated properties).

T2-2: *What is the street of [Cinema_name]?*
T2-3: *What is the postal code of [Cinema_name]?*
T2-4: *Where is the destination of [Cinema_name]?*
T2-5: *What is the postal address of [Cinema_name]?*

```
SELECT  ?streetValue  ?postalCodeValue  ?DestinationName
WHERE {
?Cinema          prefix:name   "[Cinema_name]"^^<xsd:string>.
?Cinema          prefix:hasPostalAddress         ?PostalAddress.
?PostalAddress   prefix:street              ?streetValue.
?PostalAddress   prefix:postalCode          ?postalCodeValue.
?PostalAddress   prefix:isInDestination     ?Destination.
?Destination     prefix:name                ?DestinationName.   }
```

For the main class MovieShow which has no 'name' property, we located the 'hasEventContent' property first and took each of the other three associated properties (*isInSite*, *hasPrice*, *hasPeriod*) to create one-constraint predictive questions. Since the four properties are all object properties, their range classes can be zoomed in to expand more associated properties, which results in more possible questions. Here, we only selected some of them, e.g. *Movie:name*, *Cinema:name*, *TicketPrice:priceValue,* to create reasonable predictive questions, for example:

T1-5: *What is the name of the movie which is shown in [Cinema_name]?*
T1-6: *What is the name of the movie which has the ticket price [priveValue_value]?*
T1-7: *What is the name of the movie which is shown at [startTime_value]?*
T1-8: *What is the name of the movie which is shown on [startDate_value]?*
T2-6: *What is the cinema name which shows [Movie_name]?*
T2-7: *What is the ticket price of [Movie_name]*
T2-8: *What is the show time of [Movie_name]?*
T2-9: *What is the show date of [Movie_name]?*

Predictive questions and their query templates with more constraints (e.g. two and three) can be generated as well using the same method. For generating each n-constraint question, *n* different properties need to be taken, in addition to the most frequently asked one which was located first (e.g. *name*, *hasEventContent*).

After the generation of question patterns (i.e. predictive questions) as well as the corresponding query templates, a textual entailment engine is used to find out which predictive question is entailed by a new question. Since the predictive questions and query templates contain unfilled slots, simple question processing needs to be done to identify named entities from user questions. For example, since the user question "*Where can I see the movie 300?*" is deemed to entail the predictive question "*T2-6*", we can take the SPARQL query template of the selected question to retrieve the answers to the user question. But it is necessary first to identify the fact that "*300*" is a movie name and fill in the slot in the query template with this value to produce a complete query.

5. Evaluation

This section presents a small evaluation which assesses the quality of the natural language predictive questions generated on the basis of the ontology. For the evaluation it was not necessary to assess whether or not the answers retrieved by our QA system are correct. Instead we measured to what extent our textual entailment engine was able to select the correct predictive question and as a result the correct SPARQL query template. The reason is that since the restricted-domain QA systems usually take answers from structured knowledge bases, the correct extraction of an answer depends on the generation of the correct query and not on the actual extraction process.

In this evaluation, we limited the scope of the generated predictive questions to the sub-domain of "Movies & Cinemas". For all the classes (concepts) present in this part of the ontology (shown in Figure 1), one-constraint and two-constraint questions with the corresponding SPARQL query templates were generated using the method presented in Section 4. For the textual entailment engine, we applied a bag-of-words method enhanced with the information about the entities which occur in an input user question.

The evaluation was based on 250 user questions. These questions were randomly selected from a total of 4501 benchmark questions, and include various sub-domains of the tourism domain, not only those referring to "Movies & Cinemas". Each of the user questions was marked manually by assigning to it a predictive question which is considered to be entailed by it from the machine-generated set. Some of the questions did not refer to movies or cinemas and for this reason they were marked as *outside the domain*. For a number of questions within the domain, it was impossible to find any predictive question which is entailed by the user question due to the fact that the predictive questions cannot cover all user questions with one or two constraints or the user questions contain three or more constraints. These questions were marked as *no predictive question* accordingly. The remaining questions which belong to the question type T1 and T2 with one or two constraints and are also related to movies and cinemas should have a corresponding predictive question in our set and are thus marked as *to be resolved*.

The entailment engine used in our experiment relies on two threshold values to determine whether a question is *outside the domain* or has *no predictive questions* in our set. After conducting various experiments, it was determined that if the entailment score is below *0.4,* the question is deemed *outside the domain*, whereas if the entailment score is between *0.4* and *0.5,* there are *no predictive questions* for this question. Table 1 presents a breakdown of the different categories of user questions in the evaluation and the entailment accuracy for each category.

As can be seen in the table, the results are rather low. An analysis of the results revealed several explanations, especially for the questions for which a predictive question is expected to be found (the *To be resolved* column). A first source of errors is the simplicity and similarity of the generated predictive questions. Some predictive questions are very similar which makes it difficult to differentiate between them using our entailment engine. For example, for the user question "*Who directed 300?* ", the three predictive questions, "*Who is the star of [Movie_name]?* ", "*Who is the writer of [Movie_name]?* ", and "*Who is the director of [Movie_name]?* ", obtained the same score due to the fact that the entailment engine fails to identify a link between *directed* and *director*.

Table 1. Entailment accuracy for each category of user question

Number of the user questions	Outside the domain	No predictive question	To be resolved
Total	170	24	56
Correct	122	9	25
Percentage	71.76%	37.5%	44.64%

Another source of errors is due to the very different lengths of the user questions and predictive questions. The predictive questions are quite brief since they were produced by the machine. On the other hand, the user questions produced by humans are longer and in some cases quite verbose. For this reason, there are cases where the entailment score is very low, resulting in the wrong classification of the user questions as *no predictive question* or *outside the domain*.

6. Discussion and Conclusion

This paper describes an automatic question pattern generation method for ontology-based QA, which is mainly based on ontology parsing and question entailment. Using ontology parsing, a set of question patterns can be produced together with the corresponding query templates for answer retrieval. Textual entailment is used to answer new questions by finding out which predictive questions are entailed by the new question, in this way also finding out how to answer them. This approach is very appropriate for ontology-based question answering in restricted domains because not only the question patterns, but also the query templates which in turn can be used to retrieve answers, are able to be generated on the basis of a domain ontology.

The above approach still has some drawbacks. The first is that the generated question patterns cannot cover all kinds of user questions. Especially for a large domain ontology which involves too many classes and properties, it is difficult to exhaust various combinations of properties and generate all possible question patterns. In this study, we only focus on the 'name' property to create two types of question patterns, T1 and T2. The second drawback is that some of generated question patterns do not make sense. For example, there are questions which ask for the name of a cinema at a particular set of GPS coordinates. These questions are theoretically possible, but a human would never ask them, but this over-generation does not result in a big problem.

The evaluation revealed that a large number of errors are due to the simplicity of the method used to produce the predictive questions. In light of this, it would be interesting to generate more complicated predictive questions in future, which are more similar to those produced by humans. The evaluation also revealed the limitations of a bag-of-words entailment method. Therefore, in future, we plan to experiment with alternative entailment engines which can better deal with the variability of language.

Acknowledgements

This work is supported by the EU-funded project QALL-ME (FP6 IST-033860).

References

Atzeni, P., Basili, R., Hansen, D. H., Missier, P., Paggio, P., Pazienza, M. T., and Zanzotto, F. M. 2004. Ontology-based Question Answering in a Federation of University Sites: the MOSES Case Study. In *Proceedings of the Ninth International Conference on Applications of Natural Language to Information Systems, 413-420*. Heidelberg, Berlin: Springer.

Basili, R., Hansen, D. H., Paggio, P., Pazienza M. T., and Zanzotto F. M. 2004. Ontological Resources and Question Answering. In Proceedings of the Workshop on Pragmatics of Question Answering held in conjunction with HLT-NAACL 2004. Morristown, NJ: ACL.

Harabagiu, S., and Hickl, A. 2006. Methods for Using Textual Entailment in Open-Domain Question Answering. In Proceedings of the 21st International Conference on Computational Linguistics and the 44th Annual meeting of the ACL, 905-912. Morristown, NJ: ACL.

Katz, B., Felshin, S., Yuret, D., Ibrahim, A., Lin, J., Marton, G., McFarland, A. J., and Temelkuran, B. 2002. Omnibase: Uniform Access to Heterogeneous Data for Question Answering. In *Proceeding of the Seventh International Workshop on Applications of Natural Language to Information Systems, 230-234*. Heidelberg, Berlin: Springer.

Kouylekov, M., Negri, M., Magnini, B., and Coppola B. 2006. Towards Entailment-Based Question Answering: ITC-irst at CLEF 2006. In *Proceedings of Cross Language Evaluation Forum, 526-536*. Heidelberg, Berlin: Springer.

Lopez, V., Pasin, M., and Motta, E. 2005. AquaLog: An Ontology-Portable Question Answering System for the Semantic Web. In *Proceedings of the Second European Semantic Web Conference, 546-562*. Heidelberg, Berlin: Springer.

Popescu, A., Etzioni, O., and Kautz, H. 2003. Towards a Theory of Natural Language Interfaces to Databases. In Proceedings of the Eight International Conference on Intelligent User Interfaces, 149-157. New York, NY: ACM.

Warren, D., and Pereira, F. 1982. An Efficient Easily Portable System for Interpreting Natural Language Queries. *Computational Linguistics* 8(3-4): 110-122.