An Ontology-Driven Question Answering System (AQUA)

Maria Vargas-Vera, Enrico Motta and John Domingue

Knowledge Media Institute (KMi),
The Open University,
Walton Hall, Milton Keynes, MK7 6AA, United Kingdom
{m.vargas-vera, e.motta, j.b.domingue}@open.ac.uk

Abstract. This paper describes AQUA, an experimental question answering system. AQUA combines Natural Language processing (NLP), Ontologies, Logic, and Information Retrieval technologies in a uniform framework. AQUA makes intensive use of an ontology in several parts of the question answering system. The ontology is used in the refinement of the initial query, the reasoning process (a generalization/specialization process using classes and subclasses from the ontology), and in the novel similarity algorithm. The similarity algorithm, is a key feature of AQUA. It is use to find similarities between relations/concepts in the translated query and relations/concepts in the ontological structures. The similarities detected then allow the interchange of concepts or relations in a logic formulae corresponding to the user query.

1 Introduction

In recent years, the rise in popularity of the web has created a demand for services which help users to skip over all irrelevant information quickly. One of the services is question answering (QA), the technique of providing precise answers to specific questions. Given a question such as "which country had the highest inflation rate in 2002?" a keyword-based search engines such as Google might present the user with web pages from the Financial Times, whereas a QA system would attempt to directly answer the question with the name of a country.

On the web, a typical example of a QA system is Jeeves[1] [1] which allows users to ask questions in natural language. It looks up the user's question in its own database and returns the list of matching questions which it knows how to answer. The user then selects the most appropriate entry in the list. However, users would usually prefer a precise answer to a precise question. Therefore, a reasonable aim for an automatic system is to provide textual answers instead of a set of documents. In this paper we present AQUA a question answering system which amalgamates Natural Language Processing (NLP), Logic, Ontologies and Information Retrieval techniques to provide answers to queries in a specific domain.

[1] http://www.ask.com/

The first instantiation of our ontology-driven Question Answering System, AQUA, is designed to answer questions about academic people and organizations. However, an important future target application of AQUA would be to answer questions posed within company intra-nets; for example, giving AQUA an ontology of computer systems might allow it to be used for trouble-shooting or configuration of computer systems.

AQUA is also designed to play an important role in the Semantic Web[2]. One of the goals of the Semantic Web is the ability to annotate web resources with semantic content. These annotations can then be used by a reasoning system to provide intelligent services to users. AQUA would be able to perform incremental markup of home pages with semantic content. These annotations can be written in RDF [20, 13] or RDFS [5], notations which provide a basic framework for expressing meta-data on the web. We envision that AQUA can perform the markup concurrently with looking for answers, that is, AQUA can annotate pages as it finds them. In this way, then semantically annotated web pages can be cached to reduce search and processing costs.

The main contribution of AQUA is the intensive use of an ontology in several parts of the question answering system. The ontology is used

- in the refinement of the initial query
- in the reasoning process (a generalization/specialization process using classes and subclasses from the ontology), and
- in the (novel) similarity algorithm.

The last of these, the similarity algorithm, is a key feature of AQUA. It is use to find similarities between relations/concepts in the translated query and relations/concepts in the ontological structures. The similarities detected then allow the interchange of concepts or relations in the (many-sorted) logic formulae. The ontology is used to provide an intelligent reformulation of the question, with the intent to reduce the chances of failure to answer the question.

In this paper, we describe the overall workings of AQUA. Briefly, queries formulated in plain English are translated by AQUA into many sorted logic formulae using the grammatical components obtained by our parser. AQUA then looks for an answer in different resources such as databases, a populated ontology (or knowledge base) and the Web. Currently, AQUA makes use of an inference engine which is based on the Resolution algorithm. AQUA uses also our similarity algorithm which is based on both the ontological structures and instances of the ontology, a WordNet thesaurus and the Dice coefficient. Finally, AQUA also has facilities for analyzing and explaining proofs. The explanation is provided both in pseudo-natural language and as visualization.

Our current work is focusing on embedding AQUA in the AKT semantic portal which will offer a variety of services to support academics in knowledge intensive tasks. This semantic portal offers services to users such as semantic browsing, question answering and ontological browsing. Hence, throughout

[2] The goal of the Semantic Web is to help users or software agents to organize, locate and process content on the WWW.

the paper we will illustrate AQUA using queries based on the AKT reference ontology[3].

The paper is organized as follows: Section 2 describes the AQUA process model. Section 3 describes the Query Logic Language (QLL) used in the translation of the English written questions. Section 4 describes the question classification module. Section 5 presents the query satisfaction-algorithm used in our question answering system. Section 6 describes the similarity algorithm embedded in AQUA. Section 7 describes the failure-analysis component of our question answering system. Section 8 shows output enhancements of the AQUA system. Section 9 describes a section of related work. Finally, Section 10 gives conclusions and directions for future work.

2 AQUA process model

The AQUA process model generalizes other approaches by providing a uniform framework which integrates NLP, Logic, Ontologies and information retrieval. Within this work we have focused on creating a process model for the AQUA system. Figure 1 shows the architecture of our AQUA system.

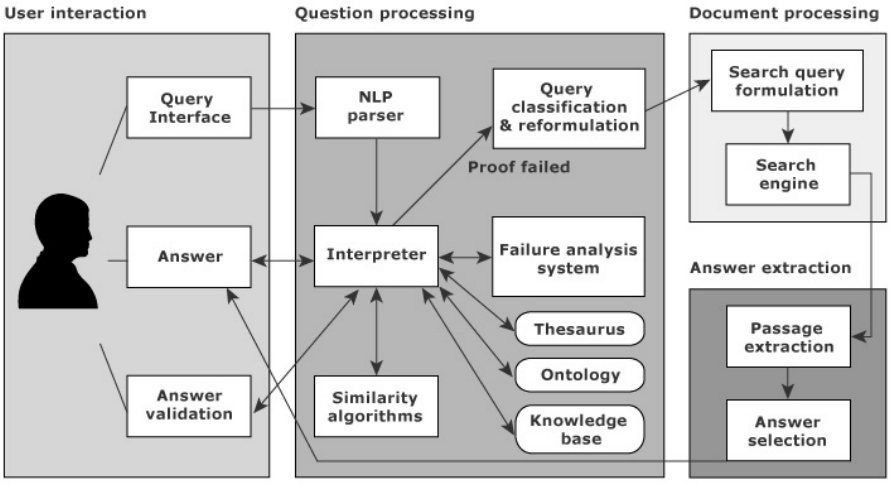

Fig. 1. The AQUA architecture

[3] The AKT ontology contains classes and instances of people, organizations, research areas, publications, technologies and events (http://akt.open.ac.uk/ocml/domains/akt-support-ontology/)

In the process model there are four phases: *user interaction*, *question processing*, *document processing* and *answer extraction*.

1. **User interaction**. The user inputs the question and validates the answer (indicates whether it is correct or not). This phase uses the following components:
   - *Query interface*. The user inputs a question (in English) using the user interface (a simple dialogue box). The user can reformulate the query if the answer is not satisfactory.
   - *Answer*. A ranked set of answers is presented to the user.
   - *Answer validation*. The user gives feedback to AQUA by indicating agreement or disagreement with the answer.
2. **Question processing**. Question processing is performed in order to understand the question asked by the user. This "understanding" of the question requires several steps such as parsing the question, representation of the question and classification. The question processing phase uses the following components:
   - *NLP parser*. This segments the sentence into subject, verb, prepositional phrases, adjectives and objects. The output of this module is the logic representation of the query.
   - *Interpreter*. This finds a logical proof of the query over the knowledge base using unification and resolution algorithms.
   - *WordNet/Thesaurus*: AQUA's lexical resource.
   - *Ontology*. This currently contains people, organizations, research areas, projects, publications, technologies and events. Currently, this is built and populated by hand.
   - *Failure-analysis system*. This analyzes the failure of a given question and gives an explanation of why the query failed. Then the user can provide new information for the pending proof, and the proof can be re-started. This process can be repeated as needed.
   - *Question classification & reformulation*. This classifies questions as belonging to any of the types supported in AQUA, (*what*, *who*, *when*, *which*, *why* and *where*). This classification is only performed if proof failed. AQUA then tries to use information retrieval approach. This means that AQUA has to perform document processing and answer extraction phases.
3. **Document Processing**. A set of documents are selected and a set of paragraphs are extracted. This relies on the identification of the focus[4] of the question. Document processing consists of two components:
   - *Search query formulation*. This transforms the original question, Q, using transformation rules into a new question Q'. Synonymous words can be used, punctuation symbols are removed, and words are stemmed.

[4] Focus is a word or a sequence of words which defines the question and disambiguates it in the sense that it indicates what the question is looking for.

- *Search engine.* This searches the web for a set of documents using a set of keywords.

4. **Answer processing.** In this phase answers are extracted from passages and given a score, using the two components:
   - *Passage selection.* This extracts passages from the set of documents likely to have the answer
   - *Answer selection.* This clusters answers, scores answers (using a voting model), and lastly obtains a final ballot.

3 Query Logic Language (QLL)

In this section we present the Query Logic Language (QLL) used within AQUA for the translation from of the English question into its Logic form. In QLL variables and predicates are assigned types. Also, QLL allows terms (in the standard recursively-defined Prolog sense [8, 22]).

Like Prolog or OCML [25], QLL uses unification and resolution [22]. However, in the future we plan to use Contextual Resolution [28]. Given a context, AQUA could then provide interpretation for sentences containing contextually dependent constructs.

Again like Prolog QLL uses closed-world assumption. So facts that are not "provable" are regarded as "false" as opposed to"unknown". Future work needs to be done in order to provide QLL with three-valued logic. When this is done an evaluation of a predicate could produce *yes*, *no* or *unknown* as in Fril [3]. Finally, QLL handles negation as failure but it does not use cuts.

AQUA uses OLL as an inter-media language as is shown in example (section 5).

The translation rules are used when AQUA is creating the logical form of a query, i.e., from grammatical components into QLL. The set of translation rules we have devised is not intended to be complete, but it does handle all the grammatical components produced by our parser. Note that variables are denoted by strings starting with a ?, for example, $?t$. The form of the logical predicates introduced by each syntax category is described as follows:

- **Nouns** (without complement) introduce a predicate of arity 1. For example the noun capital introduces the predicate $capital(?x : type\ ?t_1)$ which restricts the type of value $?x$ to be the name of the city.
- **Nouns** (with complement) introduce a predicate of arity the number of complements plus one. The pattern for n complements is as follows:

$$pred\_name(?argument_1 : type\ ?t_1, ...,?argument_n : type\ ?t_n, \\ ?argument_{n+1} : type\ ?t_{n+1}).$$

  For example, in the question "What is the population of the UK?" the noun population is translated into the predicate:

$$\text{population}(uk :?type\ t_1, \text{?x: type } ?t_2).$$

- **Qualitative adjectives** introduce a predicate of arity 1. For example, the adjective "AKT technology" translates into

$$\text{akt\_technology}(?x : type\ ?t_1).$$

- **Quantitative adjectives** introduce a binary predicate. For example, the question "How big is London?" translates into the following predicate:

$$\text{has-size}(london : type\ ?t_1, ?t : type\ ?t_2).$$

- **Prepositions** introduce a binary predicate. The pattern is as follows: $name\_preposition(?argument_1 : type\ ?t_1, ?argument_2 : type\ t_2)$. For example, the preposition *between* gets translated in the predicate:

$$\text{between}(?x : type\ t_1, ?y : type\ t_2).$$

- **Verbs** introduce predicates with one or more arguments. The first argument should be the subject of the verb, the second is the direct object, the third is the indirect object (if any) and complements (if any). For example, "David Brown visited KMi?" is translated in the following predicate:

$$visited(david\_brown : type\ ?t_1\ ,\ kmi : type\ ?t_2).$$

A set of built-in predicates is also available in QLL. This set of built-in predicates contains length, max, min, greater_than, less_than, etc.

4 Query Classification

Query classification is performed in the case that the proof fails, and gives information about the kind of answer AQUA should expect. Therefore, query classification plays an important role in the case that the answer needs to be found from the Web, and cannot be satisfied using just the populated ontology. For example, for the query "What is the capital of Mexico?" AQUA will not find anything from the ontology, and will instead generate keyword search and pass it to the search engine.

The classification phase involves processing the query to identify the category of answer that the user is seeking. The classification is performed using the information obtained during the segmentation of the sentence. During the segmentation the system finds nouns, verbs, prepositions and adjectives. The categories of possible answers are listed below.

- what/which - this kind of question appears with a head noun that describes the category of the entity involved. For this category the head noun is extracted and WordNet is used to perform the mapping between head noun and category.
- who, whom - the category of the answer is 'person'.
- when - the category of the answer is 'date'.

– why - the category of the answer should be 'reason'.
– where - the category of the answer is 'location'.

Figure 2 shows the question classification with typical question phrases in each of the 6 universal types described above.

Question Type	Question Phrases
what	"what is " "what is a"
who	"who is" "who was"
when	"when does" "when did"
which	"which is"
why	"why is"
where	"where is"

Fig. 2. Question types

The range of answers provided by AQUA our question answering system varies from yes/no answers, true/false questions, and questions which could be answered with a word or a sentence. In some cases, questions could have more than one answer, whilst in other cases the system might not find the answer.

5 The AQUA query-satisfaction algorithm

This section presents the main algorithm implemented in the AQUA system. For the sake of space, we present a condensed version of our algorithm. In this following algorithm AQUA uses steps 1-4.1 to evaluate query over the populated AKT reference ontology. Steps 4.2 to 5 are used by AQUA trying to satisfy the query using the Web as resource.

1. Parse the question into its grammatical components such as subject, verb, prepositions, object and so on.
2. Use the ontology to convert from the sorted language of QLL to a standard predicate logic. The ontology is used by a pattern-matching algorithm [5] to instantiate type variables, and allow them to be replaced with unary predicates.

[5] The pattern-matching algorithm tries to find an exact match with names in the ontology.

3. Re-write the logic formulae using our similarity algorithm (described in the next section)
4. Evaluate/execute the re-written logic formulae over the knowledge base.
   - 4.1 **If** the logic formulae is satisfied **then** use it to provide an answer
   - 4.2 **else**
     - Classify the question as one of the following types:
       * **what** - specification of objects, activity definition
       * **who** - person specification
       * **when** - date
       * **which** - specification of objects, attributes
       * **why** - justification of reasons
       * **where** - geographical location
     - Transform the query **Q** into a new query **Q'** using the important keywords.
     - Launch a search engine such as Google [6] with the new question **Q'**. AQUA will try to satisfy the user query using other resources such as the Web.
     - Analyze retrieved documents which satisfy the query **Q'**.
     - Perform passage extraction
     - Perform answer selection
     - Send answer to user for validation

The example below shows just one aspect of use of ontology in question answering: 'ontology traversal' (i.e. generalization/specialization). Suppose we have the question: "Which technologies are used in AKT?" AQUA tries to answer question as follows.

The English question is translated into the QLL expression

use(?x : type technology, akt : type ?y)

which is then converted to the standard (Prolog-style) expression by using the AKT ontology

$\exists\ X : Domain$ technology(X) $\wedge$ project(akt) $\wedge$ use(X,akt).

Where the meaning of the quantifier is determined by the type of the variable it binds. For our example, let us decide that *Domain* is the set containing the union of of each of the following congruence classes: people, projects, organizations, research areas, technologies, publications and events.

If there is a technology in the AKT project which is defined in the knowledge base then X will be bound to the name of the technology. Let us imagine the scenario where instances of technology are not defined in the AKT reference ontology. However, AQUA found in the AKT reference ontology that the relation "commercial_technology" is a subclass of "technology". Then commercial_technology is a particular kind of technology, i.e.

[6] http://www.google.com

$commercial\_technology \subseteq technology$

By using the subsumption relation our initial formulae is transformed into the following one:

$\exists\ X : commercial\_technology$ commercial_technology(X) $\wedge$ project(akt) $\wedge$ use(X,akt).

AQUA then tries to re-satisfy the new question over the knowledge base. This time the question succeeds with X instantiated to the name of one of the AKT technologies.

We can see from the algorithm that AQUA tries to satisfy a user query using several resources. Future implementations of AQUA could benefit from using the results obtained by the Armadillo[7] [7] information extraction engine running in the background as a complementary knowledge harvester. For instance, the user could ask the question What publications has Yorik Wilks produced? This is a good example of a query for which AQUA could make a request to Armadillo. Armadillo would find the publications of Yorik Wilks (for example, using CiteSeer[8] or his personal web site). Next, Armadillo would parse documents, retrieve the requested information and pass the set of publications back to AQUA to render an appropriate answer.

6 Concept and relation similarity algorithm

The success of the attempt to satisfy a query depends on the existence of a good mapping between the names of relations used in the query and names of relations used in the knowledge base. Therefore, we have embedded in AQUA a similarity algorithm. Our similarity algorithm uses both ontological structures and instances in the selected ontology, the Dice coefficient[9] [11, 23] and the WordNet thesaurus. Our similarity algorithm differs from other similarity algorithms in that it uses ontological structures and also instances. Instances provide evidential information about the concepts/relations being analyzed. In short our similarity algorithm, in order to assess similarity between terms, relies on the analysis of the neighborhood of concepts in the ontology. A relation between

[7] Armadillo is an information extraction engine which uses resources such as CiteSeer to find a limited range of information types such as publications.

[8] http://citeseer.nj.nec.com/cs

[9] As a reminder, the Dice coefficient measuring the similarity of two vectors X, and Y is defined by

$$\frac{2|X \cap Y|}{|X| + |Y|}$$

It is normalized for length by dividing by the total number of non-zero entries, and doubled so that we get a measure that ranges from 0.0 to 1.0. A coefficient of 1.0 indicates identical vectors.

two concepts provide less information about context of these concepts if they are highly connected in the original graphs and gives more information when they are weakly connected.

Our similarity algorithm uses a graph containing a subset of the ontology (relevant to the query) and the graph obtained from the question. The output is the degree of similarity and the relation name.

For the sake of space, we present a condensed version of our algorithm which shows only the similarity between relations. We show the case when both arguments, on the user query, match exactly with terms in the AKT reference ontology. Note that the similarity between concepts is not presented in this paper.

The similarity algorithm is defined as follows:

$\underline{SimilarityBase(P, \theta)}$

INPUTS:

- P the QLL expression obtained from the parser
- θ a number used to set a lower threshold for similarity

OUTPUTS:

- *ontological_relation* a replacement relation for the name of the relation in the logic formulae of the query
  a value of null_string will be used to indicate failure
- two graphs G_1 and G_2
- S_c the concept similarity [10]
- S_r the relation similarity [11] .

Procedure:

1. Build a graph for the question using the QLL expression. Call this graph (G_1). The ontology is used to instantiate type variables in the QLL expression.
2. Obtain from the ontology a sub-hierarchy in which grounded [12] arguments/ concepts from the user question can be found (i.e., to find in the selected ontology a neighborhood containing the grounded arguments). Call this graph G_2. In building G_2 we select the most general class which should appear in graph G_2. This process is done by using a hand-crafted dictionary containing *domain* information about each relations/arguments in the AKT ontology.
3. Find the intersection [13], G_c, of G_1 and G_2 based upon the node labels.
4. Compute the similarity between G_1 and G_2 using the Dice coefficient.

[10] S_c indicates how many concepts G_1 and G_2 have in common.
[11] S_r indicates how similar the relation between the same concepts are.
[12] grounded argument means instantiated argument.
[13] Intersection means to find a subgraph in G_2 which contains all concepts contained in graph G_1.

- Concept similarity is computed as follows:

$$S_c = \begin{cases} 0 & \text{no common concepts} \\ 1 & \text{same set of concepts} \\ \frac{n(G_c)}{n(G_1)+n(G_2)} & \text{otherwise} \end{cases}$$

  where n(G) is the number of concept nodes of a graph G
- Relation similarity is computed as follows:

$$S_r = \left\{ \frac{m(G_c)}{mNg(G_1)+mNg(G_2)} \right.$$

  where $m(G_c)$ is the number of arcs
  $mNg(G_c)$ is the number of arcs in the immediate neighborhood of the graph G_c

5. **If** ($S_r > \theta$ and $S_c > 0$) **then**
   - Use the graph G_2 (subset of the selected ontology) to obtain a list of ancestors of node Arg1 (L_1 = List_ancestors_Arg1) and a list of ancestors of node Arg2 (L_2 = List_ancestors_Arg2) where ancestor means the father of a node in a graph. *Arg1* and *Arg2* are arguments of the logic form of the user's query (i.e. relation-name(Arg1, Arg2)).
   - Find if exist an arc/relation between elements of both lists of ancestors. Let us assume that $ancestor_n$ from L_1 and $ancestor_m$ from L_2 are connected in Graph G_2.
   - Obtain name of the relation between $ancestor_n$ and $ancestor_m$ and instantiate *ontological_relation* to this name [14].
   - Return

   Else
   - *ontological_relation* = null_string
   - Return

A procedure called SimilarityTop uses the SimilarityBase algorithm (defined above), WordNet synsets, and feedback from the user. The SimilarityTop procedure checks if there is similarity between the name of the relation/concept in the query and the relation/concept in the selected ontology. If there is no similarity then it offers all the senses which are found in the WordNet thesaurus to the user. It is SimilarityTop that AQUA uses, with the selected sense, to rewrite the logic formulae. The main steps are defined as follows:

[14] If more than one relation is suggested to be similar AQUA rewrites the query and tries to generate the proof using one at the time until all alternatives are considered.

SimilarityTop procedure:

1. Call our similarity algorithm (defined above) i.e.
   $ontological\_relation = SimilarityTop(P, \theta)$
2. If ontological_relation $\neq$ null_string then
   evaluate_goal(ontological_relation(Arg1,Arg2))
3. Else
   - Obtain synsets for relation_question using WordNet thesaurus
   - Ask user to select sense from the ones that WordNet thesaurus provided
   - Call evaluate_goal(selected_sense(Arg1, Arg2))

Where evaluate_goal means call(Goal) using Prolog terminology.

To illustrate how our similarity algorithm works we present an example. Note that in more complex examples the graph G_2 could have several relations which could be assessed for similarity.

Let us imagine that someone asks the question: "Does Enrico Motta work on AKT?"

work(enrico-motta,akt).

By refining our query using the AKT reference ontology we obtain the following formulae:

project(akt) $\wedge$ *researcher(enrico-motta)* $\wedge$ work(enrico-motta,akt).

if AQUA evaluates this query over the knowledge base, it is likely that the question will fail. The problem is that the name of the relations in the knowledge base and the names of relations in the question might be completely different. AQUA will ask the user if they want to use similarity. If the answer to this question is "yes" then AQUA builds three graphs: the graph associated with the question (G_1), the graph using ontological structures (G_2) and the intersection graph of G_1 and G_2. These graphs are shown in figure 3.

The concept similarity and relation similarity is computed.

$$S_c = 1.0$$

$$S_r = \left\{ \frac{2m(G_1)}{mNg(G_c)+mNg(G_2)} = \frac{2(3)}{3+4} = 0.85 \right.$$

The outputs of the similarity algorithm are 1.0 and 0.85 and the name of the relation between concept *researcher* and concept *project* is "involved-in", i.e. $name\_relation(researcher, project)$ = involved-in.

The question is re-written as follows:

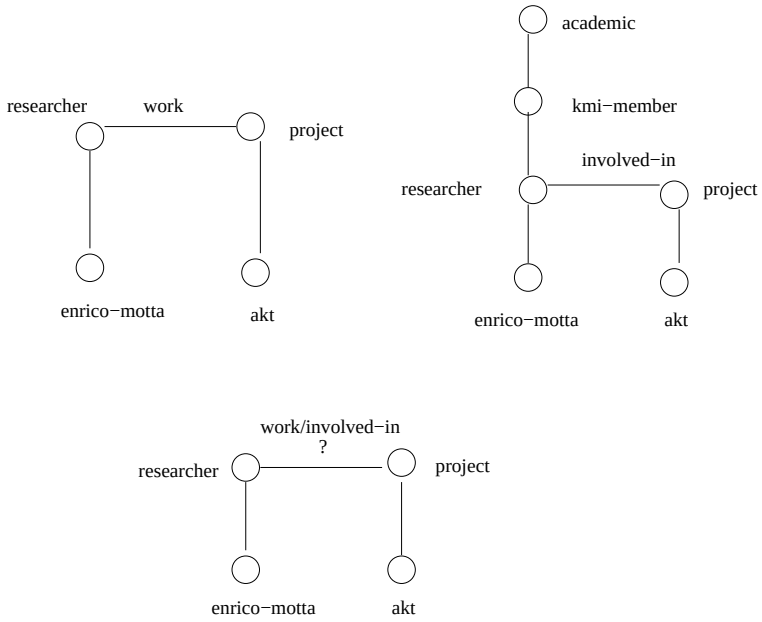

Fig. 3. Graph G_1 and G_2 and intersection G_c

project(akt) $\wedge$ *researcher(enrico-motta)* $\wedge$ involved-in(enrico-motta,akt).

The question is re-evaluated by the interpreter. By using the similarity algorithm AQUA tries to reduce to a minimum the possibility of failure because of mismatches between relation names. Since, there are several techniques for assessing similarity ([9], [26], [29]), a future implementation of AQUA will contain several algorithms for similarity.

7 Failure Analysis

The question is evaluated over the knowledge base. If the evaluation fails then the system must provide an explanation for why the question fails? There are several reasons why a query evaluated against the knowledge base could fail:

- The **term/concept** is not defined in the knowledge base. Failure occurred because the noun in the English question was not defined in the ontology and also AQUA did not find similarity with any of the terms/concepts in the ontology.
- One or more **relations** are not defined in the ontology (i.e. the similarity algorithm did not find similar relations).
- A **predicate** evaluation fails. The reason could be that the predicate used in translation is not a built-in predicate in QLL.

When a goal fails the system keeps a record of the failed goal. This helps AQUA to produce a suitable failure message. AQUA also has facilities for analyzing and explaining proofs. The explanation is provided both in pseudo natural language and as a visualization. In this way the user might be able to re-write the English question using only entity definitions which are already defined in the ontology and predicates defined in QLL.

8 Output enhancements

Figure 4 and Figure 5 show snapshots of the AQUA system. In Figure 5 we can see that AQUA not only provides a set of names which satisfy the query (the set of X such that P(X)). AQUA enhances its answer using the information from the AKT reference ontology. It provides more information about each element in the set. For instance, it brings additional contextual information such as AKT is a project at KMi and each person of the AKT team is a researcher at KMi.

Validation of answers is a difficult task in general. Therefore, AQUA provides a visualization of proofs for supporting validation of answers. Another way, provided by AQUA, to help users in validation, could be by enhancing answers with extra information. There is ongoing work at KMi on this problem. We use Magpie [10] in this task of enhancing answers. Magpie is a semantic browser which brings information about ontological entities. For instance, it can retrieve home pages related to AKT or personal web pages of researchers working in the AKT project. All this extra information could be used by our AQUA users in answer validation.

9 Related work

There are many trends in question answering [16, 17, 18, 27, 4, 24, 14, 15, 2], however, we only describe the systems most closely related to the AQUA system philosophy.

MULDER is a web-based QA system [19] that extracts snippets called summaries and generates a list of candidate answers. However, unlike AQUA, the system does not exploit an inference mechanism, and so, for example, cannot use semantic relations from an ontology.

QUANDA is closest to AQUA in spirit and functionality. QUANDA takes questions expressed in English and attempts to provide a short and concise answer (a noun phrase or a sentence) [6]. Like AQUA, QANDA combines knowledge representation, information retrieval and natural language processing. A question is represented as a logic expression. Also knowledge representation techniques are used to represent questions and concepts However, unlike AQUA, QUANDA does not use ontological relations.

ONTOSEEK is a information retrieval system coupled with an ontology [12]. ONTOSEEK performs retrieval based on content instead of string based retrieval. The target was information retrieval with the aim of improving recall

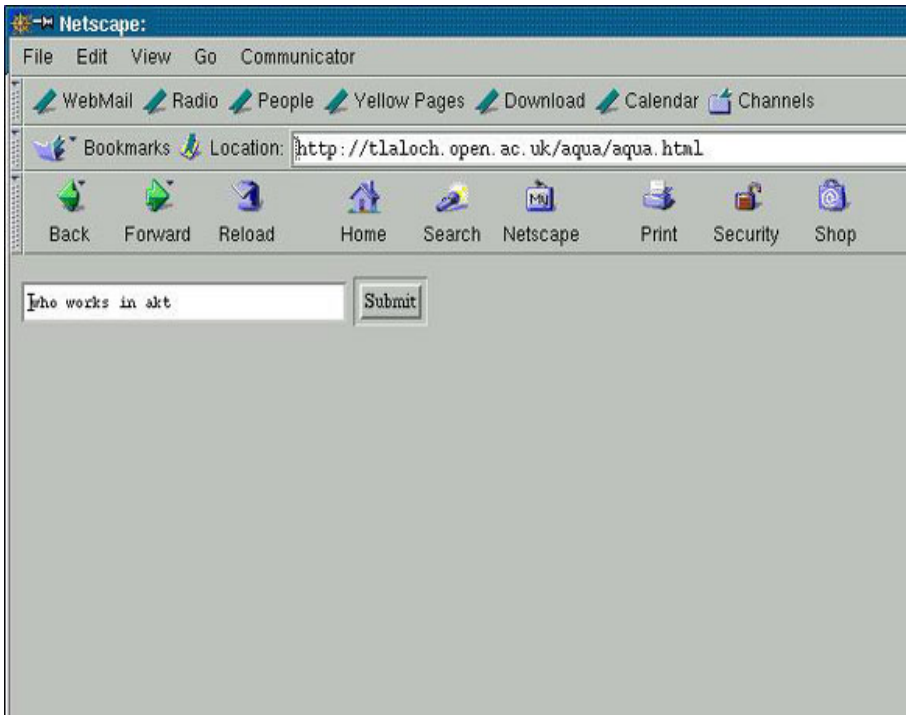

Fig. 4. Question

and precision and the focus was specific classes of information repositories: Yellow Pages and product catalogues. The ONTOSEEK system provides interactive assistance in query formulation, generalization and specialization. Queries are represented as conceptual graphs, then according to the authors "the problem is reduced to ontology-driven graph matching where individual nodes and arcs match if the ontology indicates that a subsumption relation holds between them". These graphs are not constructed automatically. The ONTOSEEK team developed a semi-automatic approach in which the user has to verify the links between different nodes in the graph via the designated user interface.

10 Conclusions and future work

In this paper we have presented AQUA - a question answering system which amalgamates NLP, Logic, Information Retrieval techniques and Ontologies [15]. AQUA translates English questions into logical queries, expressed in a language, QLL, that are then used to generate of proofs. Currently AQUA is coupled with the AKT reference ontology for the academic domain. In the future, we plan to couple AQUA with other ontologies from our repertoire of ontologies.

[15] AQUA has been implemented in Sicstus Prolog, C, OCML and PHP.

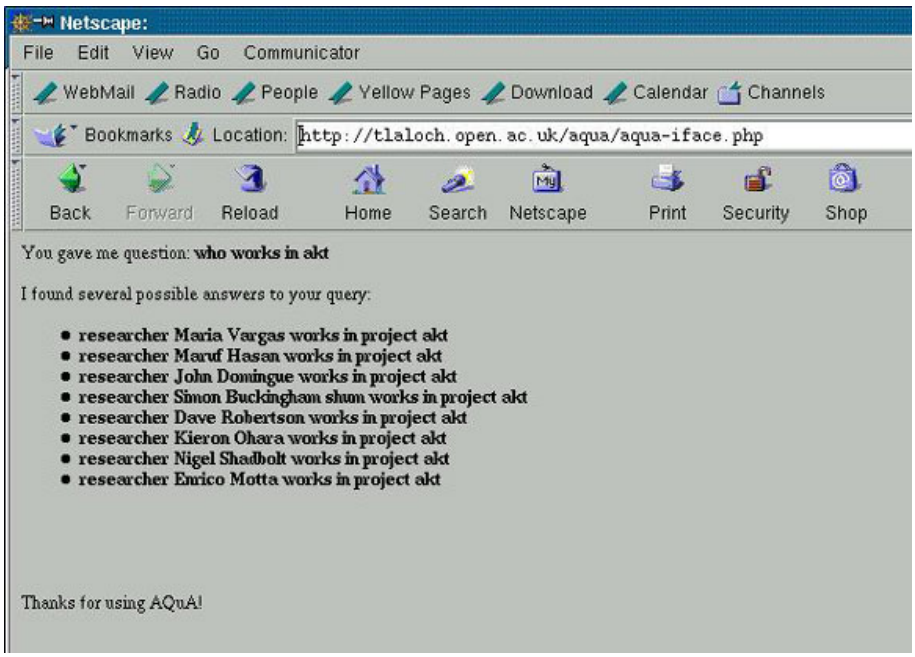

Fig. 5. Question

AQUA makes use of an inference engine which is based on the Resolution algorithm. However, in future it will be tested with the Contextual Resolution algorithm which will allow the carrying of context through several related questions.

We have also presented our similarity algorithm embedded in AQUA which uses Ontological structures, the Dice coefficient and WordNet synsets. This algorithm is used by AQUA to ensure that the question does not fail because of a mismatch between names of relations. In the future, we intend to provide AQUA with a library of similarity algorithms.

We will also explore the automatic extraction of inference rules, since knowledge about inference relations between natural language expressions is very important for the question answering problem.

11 Acknowledgments

This work was funded by the Advanced Knowledge Technologies (AKT) Interdisciplinary Research Collaboration (IRC), which is sponsored by the UK Engineering and Physical Sciences Research Council under grant number GR/N157764/01. The AKT IRC comprises the Universities of Aberdeen, Edinburgh, Sheffield, Southampton and the Open University. The authors would like to thank Arthur

Stutt and Mark Gaved for their invaluable help in reviewing the first draft of this paper. We also thank Martin Dzbor for his help on the AQUA interface and Sarah Sutcliffe for improving pictures in this paper.

References

1. Askjeeves: http://askjeeves.com/, 2000.
2. G. Attardi and A. Cisternino and F. Formica and M. Simi and A. Tommasi: Proceedings of TREC-9 Conference, NIST, pp 633-641, 2001.
3. J. F. Baldwin and T.P. Martin and B. W. Pilsworth. Fril - Fuzzy and Evidential Reasoning in Artificial Intelligence. Research Studies Press in 1995.
4. R. D. Burke and K. J. Hammond and V. A. Kulyukin and S. L. Lytinen and N. Tomuro and S. Schoenberg: Questions answering from frequently-asked question files: Experiences with the FAQ Finder System. The University of Chicago, Computer Science Department, 1997, TR-97-05.
5. D. Brickley and R. Guha:Resource Description Framework (RDF) Schema Specification 1.0. Candidate recommendation, World Web Consortium, 2000. URL:http://www.w3.org/TR/2000/CR-rdf-schema-20000327.
6. E. Breck and D. House and M. Light and I. Mani: Question Answering from Large Document Collections, AAAI Fall Symposium on Question Answering Systems, 1999.
7. F. Ciravegna and A. Dingli, D. Guthrie and Y. Wilks: Mining Web Sites Using Unsupervised Adaptive Information Extraction. Proceedings of the 10th Conference of the European Chapter of the Association for Computational Linguistic, Budapest Hungary, April 2003.
8. W. F. Clocksin and C. S. Mellish: Programming in Prolog, Springer-Verlag, 1981.
9. A. Doan and J. Madhavan and P. Domingos and A. Halevy: Learning to Map between Ontologies on the Semantic Web. In Proc. of the 11th International World Wide Web Conference (WWW2002). 2002.
10. J. Domingue and M. Dzbor and E. Motta: Semantic Layering with Magpie. Technical Report KMI-TR-125. February, 2003.
11. W. Frakes and R. Baeza-Yates: Information Retrieval: Data Structures & Algorithms, Prentice Hall, 1992.
12. N. Guarino: OntoSeek: Content-Based Acess to the Web, IEEE Intelligent Systems, pp 70-80,1999.
13. P. Hayes: RDF Model Theory, W3C Working Draft, February 2002. URL:http://www.w3.org/TR/rdf-mt/
14. E. Hovy and L. Gerber and U. Hermjakob and M. Junk and C-Y Liu: Question Answering in Webclopedia, Proceedings of TREC-9 Conference, NIST, 2001.
15. E. Hovy and L. Gerber and U. Hermjakob and C-Y Liu and D. Ravichandran: Toward Semantics-Based Answer Pinpointing, Proceedings of DARPA Human Language Technology conference (HLT), 2001.
16. B. Katz: From sentence processing to information access on the world wide web, Proceedings of AAAI Symposium on Natural Language Processing for the World Wide Web, 1997.
17. B. Katz and B. Levin: Exploiting Lexical Regularities in Designing Natural Language Systems, MIT Artificial Intelligence Laboratory, 1988, TR 1041.
18. B. Katz: Using English for Indexing and Retrieving, MIT Artificial Intelligence Laboratory, 1988, TR 1096.

19. C. Kwok and O. Etzioni and D. S. Weld: Scaling Question Answering to the Web, World Wide Web, pp 150-161, 2001.
20. O. Lassila and R. Swick: Resource Description Framework (RDF): Model and Syntax Specification. Recommendation. World Wide Web Consortium, 1999. URL: http://www.w3.org/TR/REC-rdf-syntax/.
21. D. Lin and P. Pantel: Discovery of Inference Rules for Question Answering, Journal of Natural Language Engineering", 2001.
22. J. W. Lloyd: Foundations of Logic Programming, Springer-Verlag, 1984.
23. C.D. Manning and H. Schutze: Foundations of Statistical Natural Language Processing. The MIT Press , Cambridge Massachusetts. 1999.
24. D. Moldovan and S. Harabagiu and M. Pasca and R. Mihalcea and R. Goodrum and R. Girju and V. Rus: LASSO: A Tool for Surfing the Answer Net. Proceedings of TREC-8 Conference, NIST, 1999.
25. E. Motta: Reusable Components for Knowledge Modelling. IOS Press. Netherlands, 1999.
26. N. Noy and M. Musen: PROMPT: Algorithm and Tool for Automated Ontology Merging and Alignment. In Proc. of the 17th National Conference on Artificial Intelligence (AAAI), 2000.
27. L. Plamondon and G. Lapalme and R. Diro and L Kosseim: The QUANTUM Question Answering System, Proceedings of TREC-9 Conference, NIST, 2001.
28. S. G. Pulman: Bidirectional Contextual Resolution, Computational Linguistic Vol 26/4, 497-538, 2000.
29. Z. Wu and M. Palmer: Verb semantics and Lexical Selection. 32nd Annual Meetings of the Association for Computational Linguistics. 1994.

This article was processed using the LaTeX macro package with LLNCS style