Answering N-Relation Natural Language Questions in the Commercial Domain

Elena Cabrio, Catherine Faron Zucker, Fabien Gandon, Amine Hallili, Andrea G. B. Tettamanzi

Answering N-Relation Natural Language Questions in the Commercial Domain

Elena Cabrio*, Catherine Faron Zucker†, Fabien Gandon*, Amine Hallili† and Andrea Tettamanzi†
*Inria Sophia Antipolis Méditerranée, France
Email: {fabien.gandon,elena.cabrio}@inria.fr
†Univ. Nice Sophia Antipolis, I3S, UMR 7271, France
Email: {hallili,faron,tettaman}@i3s.unice.fr

***Abstract*—This paper presents SynchroBot, a Natural Language Question Answering system in the Commercial Domain. It relies on an RDF dataset and an RDFS ontology that we have developed for the commercial domain of the mobile phone industry. We propose an approach to understand and interpret natural language questions, based on the use of regular expressions to identify both the properties connecting entities, and their values. These regex are automatically learned from a subset of our dataset with a genetic algorithm.**

I. Introduction

During the last decade, the popularity of electronic commerce (or e-commerce) has tremendously grown along with the customer turnout, leading to the emergence of a set of dedicated Business-To-Client (B2C) services and applications (e.g., Search Engines, Comparative Shopping Systems, Question Answering Systems, and Dialog Systems). At the same time, user needs are getting more and more complex and specific, especially when it comes to commercial products, with questions mostly related to their technical aspects (e.g. price, color, seller, etc.). One of the challenges to overcome to answer these needs is: *how can a system understand and interpret complex natural language (NL) questions (also known as N-relation questions) in a commercial context?*

In this paper, we answer this research question by proposing an ontology-based approach to understand and interpret complex NL questions, based on the use of regular expressions (regex) to identify both the values of properties and the properties themselves connecting entities. These regex are automatically learned from a subset of our dataset with a genetic programming approach based on the work described in [1], [2]. We implemented the whole approach in the SynchroBot NL Question Answering System.

The paper is organized as follows. Section II presents our approach of N-relation question analysis and modeling. Section III presents our approach to automatically learn property value patterns. The implementation of the overall approach in SynchroBot and its evaluation are then detailed in Section IV. Section V concludes.

II. Question Interpretation

The question interpretation step in SynchroBot relies on a standard approach, adopted by several existing systems (for a complete overview of QA systems over Linked Data, see [3]). In our use case, we move from a multiple domain Knowledge Base (KB) —like DBpedia, —to a domain-specific one for the e-commerce domain—, for which we built the Merchant Site Ontology (MSO) and the Phone Ontology (PO) [4].

Basically, to interpret a simple NL question with only one relation, the approach is to identify the key elements in the question, namely: (i) the Expected Answer Type (EAT), which is the type of the resource that we are looking for, (ii) the Named Entity (NE) representing the subject of the question, and (iii) the property, representing the relation linking the entity to the resource that we are looking for. For instance, in the question *"Who sells Nexus 5?"*, the EAT is `mso:Seller`, the NE is `msr:Nexus_5`, and the property is `mso:sells`.

A. Expected Answer Type (EAT) Detection

By default, the type of the expected answer is `mso:Product`. To improve the precision of the query to better answer the user question, the detection of the Expected Answer Type (EAT) consists of considering all the classes in the MSO ontology recognized in the user question. This is done by matching the lemma of the words in the user question with the class labels in the ontology. In our approach, the EAT is very useful for limiting the number of candidate NE and optimizing their retrieval. The EAT is used to specify the type of the queried resources during the Named Entity Recognition (see Section II-B). For example, in the question *"What is the average rate of the cell phone Lumia 1020 4G LTE?"*(id = 34, training set), the type `po:Phone` will be matched with the string `cell phone`. This means that all the retrieved resources must be phones and, as a consequence, the system will retrieve the corresponding resource `1020_4G_LTE` whose type is `po:Phone`. The EAT is used during the Graph Instantiation (see Section II-D2) to specify the type of the resource that we want to retrieve. For instance, in the question *"Which HTC cases come in warm black?"*(id = 28, test set), the system will generate a query with a variable of type `po:Case`.

B. Named Entity Recognition (NER)

The classic definition of Named Entity Recognition (NER) is to locate and classify elements in textual inputs into known categories (e.g., persons, organizations, locations, times, dates). For NER in a specific domain, all the categories of resources relevant to the domain may be of interest, and the recognition approach must be adapted to the domain. In our case, for the commercial domain relative to mobile phones, we have collected data from various data sources (e.g., relational

databases, Web services, APIs) and have constructed a KB on sellers, smartphones, and their accessories (e.g., headphones, chargers, cases). These are the categories we consider for NER. We have proposed and implemented a specific algorithm to cover this domain that we explain below.

To recognize a NE in a NL question, we match the query words with the values of three properties chosen for their relevance with respect to the domain, namely `mso:legalName`, `mso:name`, and `mso:description`. We consider the product name and description in addition to the product legal name in order to overcome product name variations and abbreviations. Commercial products in general, and smartphones in particular, are often called by pseudonyms instead of their release names, e.g., Samsung Galaxy S5 is often refered to as Samsung S5.

The recognized NEs are assigned a score as follows. The pre-selected properties whose values are used for the NER are assigned a coefficient to order them depending on their relevance. For instance, matching a word in the question to the value of the `mso:legalName` property will be more accurate than matching it to the values of the `mso:name` or `mso:description` properties, because the former contains the exact name of the product, while the latter two contain extra information (e.g., color, manufacturer, storage). Then a score is assigned to the NE according to the following strategy: the higher the number of matched words, the higher the score, and the smaller the number of retrieved resources, the higher the score. This may be formalized as follows:

$$Score(NE) = \frac{1}{\|NEs\|} \sum_{i=1}^{n} coef(PV_i) \cdot \|MWs\|, \quad (1)$$

where n is the number of matched properties, PV_i is the matched property values, $coef$ is the function weighting the properties depending on their relevance, MWs is the set of matched words, and NEs is the set of recognized NE.

C. Property Identification

In this section, we present the two methods we combine for property identification. The first method consists of capturing occurrences (labels) that have the same meaning as the properties in our ontology. The second one consists of detecting the property and its value by using specific regular expressions.

1) Label-based Property Identification: All the classes and properties of our ontology are associated with labels. This makes it possible to detect the property involved in a question by matching the lemma identified with the labels of the properties in the ontology (we first remove stop words and perform a lemmatisation using Stanford Part of Speech Tagger). Our approach is similar to the one described in [5] and follows a scoring strategy allowing us to pick up the relevant properties. For instance, in the query *"Give me the details of the products cheaper than 200$"*, the lemma *"detail"* will be matched with the `mso:description` property.

This approach works fine when the user specifies the property sought for in the question. However, the user may also specify the value of the property without the property itself. For instance, in the query *"Give me the details of the products cheaper than 200$"*, property `mso:price` is implicit. Handling such queries requires a more sophisticated approach: the system must be able to detect a property based on its value. Moreover, as concerns the processing of N-relation questions, involving more than one property, property values have to be captured in the user's question and each one correctly associated to the right property.

```
mso:price rdfs:label "price"@en ;
  rdfs:subPropertyOf <http://schema.org/price> ;
  sbmo:regexExtractionPattern
  "(?i)[0-9]+([,|.][0-9]+)?(dollar(s?)|euro(s?)|
                              ...|$|...)";
  rdfs:comment "The price of a product."@en ;
  rdfs:domain mso:Product ;
  rdfs:range xsd:double ;
```

Fig. 1. Definition of property `mso:price`

2) Value-based Property Identification: To identify a property by its value when the property name is omitted in a query and to correctly associate values and properties in complex queries, we introduced in our Meta-Ontology the `sbmo:regexExtractionPattern` property, whose domain is the class of properties and whose value is a regular expression (regex) allowing to identify a property in a query by matching the regex with the detected values. Figure 1 shows the regex associated to property `mso:price`.

We first manually created a regex for each of the 28 properties of the MSO ontology after analysing its lineaments. Then we proposed a supervised method based on Genetic Programming to automatically generate regular expressions for each property in our ontology which will be described in Section III.

D. Construction of a Relational Graph

Once the properties occurring in the user query and their values are identified, the construction of a complete representation of the query requires the creation of a query *graph* connecting the identified properties. We first create a graph structure relating the identified properties, then we instantiate this graph with the detected property values.

1) Construction of the Relational Graph Structure: We use property domains and ranges to connect properties and create a relational graph as query graph pattern. Two properties sharing the same domain or the same range for Object properties ,or one having as range the domain of the other, are connected by their domains or ranges. For example, let us consider the following question *"Give me the details of the products cheaper than 200$"*. The system will identify the two properties `mso:description` and `mso:price` sharing the same domain and it will create a connected graph.

However, the resulting graph may not be connected. For instance, if the user asks *"Give me the address of the products cheaper than 200$"*, the system will detect the two properties `mso:address` and `mso:price` and will not be able to construct a connected graph with them, because the address concerns the seller of the product and not the product itself. In such cases, we look for the minimum number of properties that would make the graph connected and we search for properties with appropriate domains and range. For instance, for the

above example query, property `mso:seller` will be added in the graph because its domain and range match the domain and range of the other properties in the relational graph.

2) Instantiation of the Relational Graph: The relational graph representing the user NL query is constructed from its graph structure by replacing its nodes by the detected property values or with variables if no value is detected for some properties. This graph is formalized in the SPARQL language: it is the graph pattern of the SPARQL query that the system will execute against the KB to retrieve answers to the NL query. For instance, Figure 2 presents the SPARQL query for our running example NL query.

```
SELECT DISTINCT *
WHERE {
  ?ne a mso:Product ; mso:name ?n
  OPTIONAL {?ne mso:description ?var1}
  OPTIONAL {
    ?ne mso:price ?v. ?v rdf:value ?var2
    FILTER (CONTAINS(?var2, lcase(str("200"))))}
  BIND(IF(BOUND(?var1),1,0)+
       IF(BOUND(?var2),1,0) AS ?c) }
ORDER BY DESC (?c) LIMIT 20
```

Fig. 2. The SPARQL query representing the NL question *"Give me the details of the products cheaper than 200$"*

If several possible values are identified for one property, the system will generate a query for each value. If several possible graph structures are constructed, the system will generate a query for each of them.

III. Genetic Learning of Property Value Patterns

Our approach to learning regular expressions for property value detection is based on the work described in [1], [2], which uses genetic programming.

The idea is to derive a regular expression for the values of each property in the ontology, by using a training dataset composed of a pair of strings `(TCVTE,VTE)` where `TCVTE` is the Text Containing the Value To Extract and `VTE` is the Value To Extract. For instance, Table I presents a few examples of `(TCVTE,VTE)` pairs used to learn a regex for the values of property `mso:name`.

Text (TCVTE)	Value (VTE)
Patriot Memory - FUEL+ 5200 mAh Rechargeable Lithium-Ion Battery and Signature Series **8GB** microSDHC Memory Card	8GB
Samsung - Galaxy Mega 4G with **16GB** Memory Cell Phone - Black (AT&T)	16GB
T-Mobile Prepaid - Apple iPhone 5 with **16GB** Memory No-Contract Mobile Phone - Black & Slate	16GB
HTC - One (M7) 4G LTE with **32GB** Memory Cell Phone - Black (Sprint)	32GB
...	...
Nokia - Lumia 520 with **8GB** Memory Cell Phone (Unlocked) - Black	8GB

TABLE I. Example of pairs `(TCVTE,VTE)` to learn a regex for property `mso:name`

Each regex is represented by its syntax tree, which contains the (internal) function nodes (the central dot · stands for a subtree, possibly consisting of a single node):

- concatenation node: to concatenate two subtrees;
- possessive quantifiers: {·*+, ·++, ·?+, ·{m,n}};
- group operator: (·);
- character class operator: [·];

and the following (leaf) terminal nodes:
- constants: a character, a number or a string;
- ranges: {a-z, 0-9, a-z0-9, A-Z, A-Z0-9};
- character classes: {\w (for characters), \d (for digits)};
- white space: \s
- the wildcard character: .

The principles of the genetic learning algorithm are as follows. For each property, we start by creating an initial population of regex whose size is twice the size of the example dataset (VTE). Half of the regex population is generated based on the property values in the example dataset, by replacing numbers by \d and characters by \w. For example, "32GB", which is a phone storage value, will be turned into \d\d\w\w. The other half of the initial regex population is generated by using the ramped half-and-half method [6]. This method allows generating random regex trees with different depth after setting up a ramped size. The evolutionary algorithm is then launched several times depending on a predefined number of iterations (in our case 150 iterations).

At each iteration, the generated regex are tested against the property values in the dataset and their performances are measured by computing their precision and Matthews Correlation Coefficient (MCC), defined as

$$MCC = \frac{TP \cdot TN - FP \cdot FN}{\sqrt{(TP+FP)(TP+FN)(TN+FP)(TN+FN)}}, \quad (2)$$

where TP, TN, FP, FN are the number of True Positives, True Negatives, False Positives and False Negatives.

A new population is created by using the fitness-proportionate selection method, also known as the roulette-wheel selection. According to this method, the probability for the i-th individual in the population of being selected for the next iteration is

$$p_i = \frac{f_i}{\sum_{j=1}^{N} f_i} \quad (3)$$

where f_i is the fitness of the ith individual, regex_i, computed as its MCC (cf. Equation 2) and N is the number of individuals in the regex population. This function allows to select one regex at a time until reaching the size of the new population. As a result, a population of the same size is constructed with the best scoring regex eventually occurring several times.

In this new population, regex_n and regex_{n+1} are recombined with crossover with a probability of $\frac{1}{2}$. As a result, a new population of regex having the same size of the previous one is generated and the next iteration of the learning algorithm can start. The algorithm stops when achieving a satisfying regex population (the precision of one regex equals to 1) or at the end of the allotted iterations. If a precision = 1 is achieved, then the corresponding regex is automatically selected and used. Otherwise, if the precision is greater than 80%, then the regex with the highest score is selected. If not, the learning algorithm is relaunched with a new initial population.

IV. Evaluation

A. Dataset and Protocol

We tested both the algorithm for learning property values and the algorithm for answering NL questions on the QALM [4] benchmark. QALM stands for Question Answering Linked Merchant data. We built this benchmark for evaluating QA systems that use commercial data. It provides an RDF dataset along with an ontology describing a part of the commercial domain and aligned to Schema.org ontology and a set of 70 NL questions annotated with the corresponding SPARQL queries and the correct answers retrieved from the dataset.

For the property value learning algorithm, we have extracted from the QALM dataset 500 examples of values for each property as the training set. Then we launched the algorithm on the whole dataset in the QALM benchmark. As for the NL question answering algorithm, we used the provided training set, which contains 40 NL questions, to train the system and then we tested it using the gold standard.

B. Evaluation of the Algorithm for Learning Property Values

Table II presents the results of the algorithm for learning property values. It clearly appears that the precision of the learned regex vary a lot depending on the property. The results of the algorithm are very good for properties with an infinite domain of values like `mso:price` or `mso:storage` and very bad for properties with a finite domain of discrete values like `mso:model` or `mso:color`. For these properties, we manually generated a regex with a high precision to improve the performances of the QA algorithm using these regex. Table III presents the precision of the algorithm.

Property	Learned Regex	Precision	Manual Regex	Precision
storage	[0-9]++G[a-zA-Z]	100%	\d++[Gg][Bb]	100%
price	\d++.\d++\D	97,33%	(?i)[0—9]+([,.][0—9]+) ?(euro(s?)|dollar(s?)|...|£|€|$ |...)	100%
release date	\d\d\W[0—9] ++\D\d?+	60%	\((19|20)\d\d)[—/](0? [1—9]|1[012])[—/](0? [1—9]|[12][0—9]|3[01])	100%
...	...	...	...	...
model	[0—9]++G [a—zA—Z]	20%	([A—Z]\w++)+*([A—Z]\d)	73%
color	[0—9]++G [a—zA—Z]	11%	(?i)aliceblue|antiquewhite |aqua|aquamarine|azure|beige |bisque|black ...	90%

TABLE II. Precision of the learned regex for some properties in the QALM ontology, listed in decreasing order

	Training set (500 examples)	Whole set (5000-10000)
storage	100%	100%
price	97,33%	89,6%
release date	60%	47,16%
...	...	...
model	20%	4,01 %
color	11%	3,2%

TABLE III. Precision of the learned regex on the whole QALM dataset. Properties are listed in decreasing order

C. Evaluation of the NL Question Answering Algorithm

Table IV shows the performances of the Synchrobot NL Question Answering algorithm. The current version of the system allows processing most of the questions in the dataset (58/70). It is only capable of giving 12 right answers, even if the average of output answers is relatively high. However, the proportion of partially right answers (28/58) is very encouraging in terms of enhancing the system precision.

	Whole set			Limited set		
	Precision	Recall	F-Score	Precision	Recall	F-Score
Training	0.32	0.32	0.32	0.36	0.36	0.36
Test	0.30	0.41	0.35	0.30	0.52	0.38
Average	0.31	0.36	0.33	0.33	0.44	0.37

TABLE IV. Evaluation of the NL question answering algorithm. The limited set restricts to the queries which SynchroBot is supposed to be able to answer.

A deeper analysis revealed that our system can precisely answer 1-relation questions that contain a Named Entity (up to 90% of precision). The only cases where SynchroBot fails to process 1-relation questions is when the property does not exist in the ontology. Also, the amount of answered 2-relations questions is large (63%). For N-relations and NE-less questions, the performance still needs to be improved (only 11% of precision).

V. Conclusions and Future Work

In this paper, we presented a prototype ontology-based and NL Question Answering System for the Commercial domain, aiming to answer complex queries. As ongoing work, we are currently enhancing the genetic programming algorithm for learning regex for property value extraction. Regarding the construction of the relational graph representing a NL query, we consider adopting the pattern extraction approach described in [7] in order to enhance our property identification approach.

Acknowledgements

We thank Amazon, eBay and BestBuy for contributing to this work by sharing with us public data about their commercial products. We thank the French National Association for Research and Technology (ANRT) and the SynchroNext Company which are co-funding this work.

References

[1] P. Petrovski, V. Bryl, and C. Bizer, "Learning regular expressions for the extraction of product attributes from e-commerce microdata," in *Proc. of the 2nd Int. Workshop on Linked Data for Information Extraction, LD4IE 2014, Riva del Garda, Italy*, 2014.

[2] A. Bartoli, G. Davanzo, A. D. Lorenzo, M. Mauri, E. Medvet, and E. Sorio, "Automatic generation of regular expressions from examples with genetic programming," in *Proc. of Genetic and Evolutionary Computation Conference, GECCO'12, Philadelphia, PA, USA*, 2012.

[3] V. Lopez, V. S. Uren, M. Sabou, and E. Motta, "Is question answering fit for the semantic web?: A survey," *Semantic Web*, vol. 2, no. 2, 2011.

[4] A. Hallili, E. Cabrio, and C. Faron-Zucker, "QALM: a benchmark for question answering over linked merchant websites data," in *Proc. of the ISWC 2014 Posters & Demonstrations Track, Riva del Garda, Italy*, 2014.

[5] D. Damljanovic, M. Agatonovic, and H. Cunningham, "Natural language interfaces to ontologies: Combining syntactic analysis and ontology-based lookup through the user interaction," in *Proc. of the 7th Extended Semantic Web Conference, ESWC 2010, Heraklion, Crete, Greece*, 2010.

[6] J. R. Koza, *Genetic programming - on the programming of computers by means of natural selection*. MIT Press, 1993.

[7] E. Cabrio, J. Cojan, A. Palmero Aprosio, and F. Gandon, "Natural language interaction with the web of data by mining its textual side," *Intelligenza Artificiale*, vol. 6, no. 2, 2012.