Ontology-Driven Approach to Obtain Semantically Valid Chunks for Natural Language Enabled Business Applications

Shailly Goyal, Shefali Bhat, Shailja Gulati, C Anantaram

Innovation Labs, Tata Consultancy Services Ltd
Gurgaon, India

Abstract. For a robust natural language question answering system to business applications, query interpretation is a crucial and complicated task. The complexity arises due to the inherent ambiguity in natural language which may result in multiple interpretations of the user's query. General purpose natural language (NL) parsers are also insufficient for this task because while they give syntactically correct parse, they lose on the semantics of the sentence. This is because such parsers lack domain knowledge. In the present work we address this shortcoming and describe an approach to enrich a general purpose NL parser with domain knowledge to obtain semantically valid chunks for an input query.
A part of the domain knowledge, expressed as domain ontology, along with the part-of-speech (POS) tagging is used to identify the correct predicate-object pairs. These pairs form the constraints in the query. In order to identify the semantically valid chunks of a query, we use the syntactic chunks obtained from a parser, constraints obtained by predicate-object binding, and the domain ontology. These semantically valid chunks help in understanding the intent of the input query, and assist in its answer extraction. Our approach seamlessly works across various domains, given the corresponding domain ontology is available.

1 Introduction

Natural language (NL) enabled question answering systems to business applications [1, 2] aim at providing appropriate answers to the user queries. In such systems, query interpretation is a fundamental task. However, due to the innately ambiguous nature of the natural language, interpretation of a user's query is usually not straightforward. The ambiguity can be either syntactic, e.g., prepositional phrase (PP) attachment, or it can be semantic. In order to resolve such ambiguities, NL enabled question answering systems mostly use general purpose NL parsers. Although these parsers give syntactically correct chunks for a sentence, these chunks might not be semantically meaningful in a domain. For example, consider the following queries:

- "`List the employees working in loss making projects`". In this query, a human can easily disambiguate that "`loss making`" is a modifier of "`projects`", and "`working in loss making projects`" is a modifier of "`the employees`". That is,

Special issue: Natural Language Processing and its Applications.
Research in Computing Science 46, 2010, pp. 105-116

Received 23/11/09
Accepted 16/01/10
Final version 10/03/10

the correct chunks are "[List [[the employees] [working [in [[loss making [projects]]]]]]]". However, the chunks obtained from an NL parser are "[List [[[the employees] [working [in [loss]]]] [making [projects]]]]", which will be interpreted as "List the employees who are working in loss and who make projects".

- "Give the projects having costing and billing >$25000 and <$35000, respectively". The NL chunker may chunk this query as "[Give [[the projects] [having [[costing] and [billing]] [>$25000] and [<$35000]]], respectively". From these chunks it is not possible to identify that "costing >$25,000" and "billing <$35,000" are the two constraints which modify "the projects".

Thus the chunks obtained from such parsers may not be helpful in extracting the answer to the user's query. The problem becomes even more severe in case of complex queries involving multiple constraints and nested sub-questions. Thus the problem at hand is "*How can we automatically enrich the output of a general purpose NL parser with the domain knowledge in order to obtain syntactically as well as semantically valid chunks for the queries in the domain?*"

We put forward an approach to solve the queries in a domain using the domain knowledge and the syntactic structure of the queries. A part of the domain knowledge is represented in the form of domain ontology which is based on semantic web technologies [3]. We define '*constraints*' and '*semantically valid chunks*' for a query. Semantically valid chunks aid in the interpretation and extraction of the answers to the user's queries unambiguously.

2 Related Work

Processing natural language questions to obtain an equivalent structured query has long been an area of research in artificial intelligence [2, 1]. Still, most existing approaches cannot interpret real-life natural language questions properly. Even the few which can do so depend so much on domain-specific rules, that porting to other domains becomes an issue.

Popescu et al. [4] adapts the Charniak parser [5] for domain-specific question answering by extending the training corpus of the parser with a set of 150 hand-tagged domain-specific questions. Further, semantic rules inferred from domain knowledge are used to check and correct preposition attachment and preposition ellipsis errors. START [6] decomposes complex questions syntactically or semantically to obtain sub questions that can be answered from available resources. If these answers are not sufficient to solve the question, semantic information - in the form of rules that map 'key' domain questions to the answers - is used. The main drawback of these approaches is that the creation of domain-specific rules is very resource intensive, and hence restricts portability.

AquaLog [7] tries to transform the NL question to ontology-specific triples using syntactic annotations, semantic terms and relations, and question words to interpret the natural language question. If these cannot resolve the ambiguity in the question, domain ontology and/or WordNet are used to make sense of the input query.

3 The Architecture

NL based Question Answering system requires the queries to be analyzed and chunked in an appropriate manner so as to have correct query generation and answer extraction. An NL query can be viewed as consisting of a set of *unknown predicates* whose values need to be determined based on the *constraints* imposed by the rest of the query. Domain ontology along with a POS tagger is used to identify the constraints in the query. These constraints along with the domain knowledge and the parse structure of the query are used to find the semantically valid chunk set. These chunks are then converted to a formal query language and the answer is retrieved from the ontology. Figure 1 demonstrates the overall architecture of our approach. In this figure, solid arrows represent the process flow for a query, and dashed arrows represent the information flow.

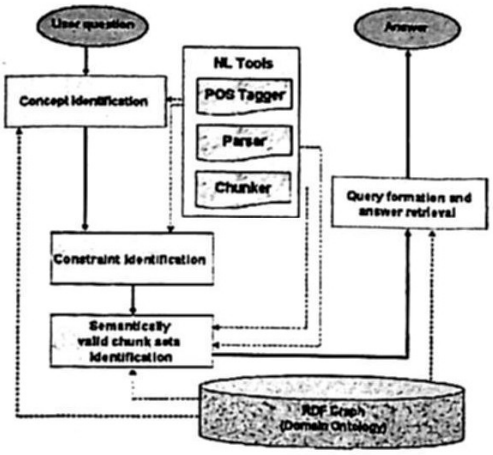

Fig. 1. The Architecture

Hence for the appropriate interpretation and analysis of the query, the important issues that need to be addressed can be summarized as:

Constraint identification. This involves identifying the correct predicate-object pairs.

Semantically valid chunk set. This involves identification of valid constraints for each unknown predicate so that correct interpretation of the given query can be ensured.

Query generation. In this step, the semantically valid chunk set is converted to appropriate formal language query using the domain ontology.

In Section 4 we briefly describe the domain ontology and formalize the terminologies used in this work. In the subsequent sections we discuss constraint identification and semantically valid chunk set formation.

4 Domain Ontology and Other Preliminaries

We use semantic web technologies [3] to create the domain ontology (in RDF format) using the relational data of the business application along with its meta information stored in the seed ontology[1] [8]. The ontology D_O of a domain D describes the domain terms and their relationships in the $\langle subject - predicate - object \rangle$ format. For illustration, $\langle$`Ritesh - project_name - Bechtel`$\rangle$ describes that the predicate '`project_name`' of the subject '`Ritesh`' has object '`Bechtel`'. A synonym dictionary having information about the synonyms of the domain terms is also maintained.

[1] The seed ontology has meta information about the domain, like $\langle ns : employee\ owl : type\ owl : person \rangle$, $\langle ns : employee\ ns : hasName\ ns : employee\_name \rangle$.

The domain ontology and synonym dictionary are used to identify the concepts in the user query Q posed in the domain D. The domain ontology D_O is used to further classify the concepts as predicates and objects. For a query Q, we denote the *set of predicates* as $P_Q = \{p_1, p_2, \ldots p_n \mid \exists\langle s - p_i - o\rangle \in D_O$, and p_i is present in the query $Q\}$. The *set of objects* present in the query Q is $O_Q = \{o_1, o_2, \ldots o_m \mid \exists\langle s - p - o_i\rangle \in D_O$ or o_i is a numerical/date value, and o_i is present in the query $Q\}$.

In this work, we have considered examples from a logical subset of the Project Management System for an organization. The database consists of tables containing data about the projects, the various costs associated with the projects, employees and their allocations in different projects. The important tables are named ProjectDetails, Employees, CostDetails and Allocations. Some of the attributes of these tables are:

ProjectDetails. project_id, project_name, project_type
Employees. employee_id, employee_name, age, gender, joining_date
CostDetails. costDetails_id, project_id, costing, billing, revenue
Allocations. allocation_id, employee_id, project_id, role

For illustration, consider the query:

Example 1. `What is the role of the associates who joined before 18/10/2008 in the SWON projects with costing and revenue more than $15000 and < $25000, respectively?`

In this query, the concepts identified using the domain ontology are: `role`, `employee_name`[2], `joining_date`, `SWON`, `project_name`, `costing`, and `revenue`. After considering the date and the numeric values in the query, i.e., `18/10/2008`, `15000` and `25000` as objects, the predicate and object set obtained are:

$P_Q = \{$`role, employee_name, joining_date, project_name, costing, revenue`$\}$,
$O_Q = \{$`SWON, 18/10/2008, 15000, 25000`$\}$.

In the following section we present an algorithm to bind these predicates and objects so as to obtain constraints for the query.

5 Constraint Identification

For a successful query creation and execution, identification and formulation of correct constraints is of utmost importance. With reference to this work, constraint identification involves binding each '*object*' in the query with its corresponding '*predicate*'. This predicate-object pair is referred to as 'constraint'.

We define a *constraint* as $c_k = (p_i, o_j), o_j \in O_Q, p_i \in P_Q$, and o_j is the value for the predicate p_i in Q. All the constraints in the query Q are identified, and $C_Q = \{c_1, c_2, \ldots c_m\}$ denotes the *constraints set.* Predicate used in any constraint is referred to as *constraint predicate.* The *set of constraint predicate* is $P_Q^C = \{p_i \mid p_i \in P_Q$ such that $\exists(p_i, o_i) \in C_Q\}$. Predicates that do not form part of the constraint set are referred to as *unknown predicates.* The *set of unknown predicates* is $P_Q^U = \{p_i \mid p_i \in P_Q, p_i \notin P_Q^C\}$.

[2] The synonym dictionary has mappings defined between '`employee`' and '`associate`' etc.

In a natural language query, constraint identification (or predicate-object binding) needs special attention due to the following reasons:

Unspecified predicates. For some (or all) of the objects present in the query, the corresponding predicate might not be explicitly specified. For example consider the query '`Give me the role of Puneet in the project having Ritesh as project leader`". Here the objects '`Puneet`' and '`Ritesh`' need to be attached to the corresponding predicate '`employee_name`', which is not specified in the query.

Constraint vs. unknown predicate. The issue of unspecified predicates becomes even more severe when a predicate p_i for an object o is present in the query, but the same predicate p_i also happens to be an unknown predicate. For example, in the query mentioned above, the value '`project leader`' in O_Q is compatible to the predicate '`role`' in P_Q. But these predicate and value are not to be bound as the predicate '`role`' is an unknown predicate, whose value needs to be determined.

Predicates followed by the respective objects. In questions with multiple constraints, sometimes predicate and its object may not be given consecutively. Instead, the query may have a predicate list followed by the corresponding object list (or vice versa). Considering the same Example 1 (Section 4), in the phrase '`with costing and revenue more than $15000 and < $25000, respectively`', the predicate list, i.e. [`costing, revenue`], is followed by the corresponding objects list, i.e. [`more than $15000, < $25000`]. We need to identify and bind the appropriate predicate-operator-object pairs from the predicate and the object lists.

In the following we describe an algorithm for predicate-object discovery and binding.

5.1 Algorithm for Predicate-Object Binding

The main steps of the algorithm are as follows.

Step 1. Operator-Object binding for numerical/date objects. The first step towards Operator-Object binding is the identification of the comparison operators in the query. For operator identification, the system maintains a mathematical operator dictionary. All the occurrences of the string comparators in the question are replaced by the corresponding mathematical comparator. Also, if there is any numeric value in the question that is not preceded by any operator, by default '=' operator is prefixed. These form the corresponding operator-object pairs. Henceforth we refer to these operator-object pairs as objects.

Step 2. Group the predicates and objects that immediately follow/preceed the POS tags of the *assignment* words[3]. POS tags of some such words are '`VBZ`', '`VBP`', '`IN`', '`SYM`' etc. We also group the predicates that are immediately followed (or preceded) by any object. In case there is a list of predicates and a list of objects satisfying the above, then these lists are also grouped. These groups are the *possible pairs* for predicate-object binding. For instance, in Example 1, we

[3] Assignment words are words which are usually specified between the predicates and its objects. E.g., form of copula 'be', preposition 'as', or mathematical operators.

get the pairs (joining_date : <18/10/2008), (project_name, (SWON)), (costing, revenue : >$15000, <$25000).

Step 3. From the groups obtained in Step 2, we bind the predicates and objects that are of the same data type. The compatibility for predicate and the object is checked using the domain ontology. In case of predicate and object list, one-on-one binding is done. For example, from the groups obtained above, we get the following predicate-object pairs: (joining_date, <18/10/2008), (costing, >$15000), and (revenue, <$25000). The pair (project_name, SWON) is not bound because the predicate of the object 'SWON' as obtained from the domain ontology is 'project_type'.

Step 4. The string objects that are not bound to any predicate in Step 3 are bound to their compatible predicates. The compatible predicate for an object is determined using the domain ontology. For instance, since the object 'SWON' is not bound to any predicate till now, it is bound to its compatible predicate 'project_type' to obtain the constraint (project_type, SWON).

The predicates bound to any object in the above steps form the constraint predicate set, and the remaining predicates constitute the unknown predicate set. Using the above algorithm for Example 1, we obtain the constraint set, constraint predicate set and the unknown predicate set as:

- C_Q = {(joining_date, <18/10/2008), (project_type, SWON), (costing, >$15000), (revenue, <$25000)}.
- P_Q^C = {joining_date, project_type, costing, revenue}.
- P_Q^U = {role, employee_name, project_name}.

The constraint sets thus obtained are used to find the semantically valid chunk set as discussed in the following section.

6 Semantically Valid Chunk Set

Semantically valid chunk set identify the conditions on each unknown predicate in the query, and are constituted from the constraints and unknown predicates as obtained in Section 5. For instance, in Example 1 (Section 4), 'joining_date < 18/10/2008' is a condition on the predicate 'employee_name'. Due to the syntactic ambiguity, more than one syntactic parse might be obtained for a NL query. Such cases may eventually result in more than one semantically viable chunk set. Formally we define semantically viable chunk sets as follows.

Definition. A *Semantically viable chunk set* (SVC set) of a query Q corresponding to the k^{th} parse is a set $SVC_{Q_k} = \{SC^p_{Q_k} \mid p \in P^U_Q\}$ where $SC^p_{Q_k}$ is a semantic chunk. *Semantic chunk* of a predicate $p \in P^U_Q$ is defined as:

- If $C_Q \neq \{\}$, $SC^p_{Q_k} = \langle p, c_1, c_2, \ldots c_i, \ldots c_r \rangle$ $(r \geq 1)$, where $c_i \in C_Q$ or $c_i = SC^{p'}_{Q_k} \in SVC_{Q_k}$, and c_i is a condition on the predicate p
- If $C_Q = \{\}$ (and $P^U_Q \neq \{\}$), $SC^p_{Q_k} = \langle p \rangle$

Such that, SVC_{Q_k} satisfies the following:

a. $\forall p \in P_Q^U, \exists SC_{Q_k}^p \in SVC_{Q_k}$.
b. $\forall c' \in C_Q, \exists SC_{Q_k}^p \in SVC_{Q_k}$ such that $SC_{Q_k}^p = (p, c_1, c_2, \ldots c', \ldots c_r)$.

The condition 'a' states that there is a semantic chunk for each unknown predicates in the query. The condition 'b' states that each constraint in the query is used in at least one semantic chunk.

Definition. For a query Q, the semantically viable chunk set which is semantically valid as per the domain ontology is the *semantically valid chunk set*, $SVaC_Q$. These sets are referred to as SVaC sets.

We can classify the queries posed for a natural language interface to business application in the following categories:

1. The unknown predicate is specified as a noun in the question. E.g., '`What is the role of Ritesh in AB Corp?`', '`Give me the project of Ritesh`'. In this case the syntactic modifiers of the predicate (noun) determine the constraints on the predicate.
2. A wh-word ('`who/when/where`') refers to the unknown predicate. E.g., '`Who is the project leader of Bechtel?`', '`When did Ritesh join Bechtel?`'
3. The unknown predicate may have a wh-word ('`what/which/whose/how much/how many`') as a determiner. In such questions the wh-word is placed before the unknown predicate, and these words ask which thing or person is being referred to. For example, '`In which project is Ritesh allocated?`', '`How many employees are allocated to Bechtel?`'
4. `Why/how` questions. These questions expect descriptive answer. Since our system aims to handle only factual questions, such questions are out of scope.
5. Yes/No questions. These questions expect '`yes`' or '`no`' as answer. Due to space limitations, such questions are kept out of scope of the current work.

We use syntactic information of the question to obtain the semantically viable chunk sets as discussed in the following section.

6.1 Finding Semantically Viable Chunk Sets

For a query, the main task for identification of semantically viable chunk sets is to identify the conditions for all the unknown predicates. We exploit syntactic information of the query for this purpose. We use a dependency-based parser (e.g. Stanford Parser [9], Link Parser [10]) to obtain the syntactic structure of the question. These parsers provide the phrase structure as well as the dependencies between different words of a given sentence. In case of syntactic ambiguity, these parsers provide all possible interpretations of the input sentence. In the following, we discuss the process of identifying the appropriate semantic chunks for different categories of queries.

Unknown Predicate as Noun. If an unknown predicate in the query plays the role of noun, its syntactic modifiers identify the constraints on the predicate. Dependency based parsers provide dependencies between noun and its modifiers. This information along with the phrase structure of the query is

used to determine the phrase modifying the unknown predicate. These phrases give the constraints for the unknown predicate. The unknown predicate with its constraint is a candidate semantic chunk. For example, for the question 'Give me the role of the associates with age > 30 years?', the preposition phrase 'with age > 30 years' is a post-nominal modifier of the noun 'associates'. The constraint corresponding to this preposition phrase is 'age > 30', and hence the corresponding semantic chunk can be obtained as

$SC_Q^{employee\_name} = \langle employee\_name, age > 30 \rangle$.

Further, the preposition phrase 'of the associates with age > 30 years' is modifying the noun 'role'. Since the semantic chunk, $SC_Q^{employee\_name}$, for the phrase 'of the associates with age > 30 years' has already been identified, the semantic chunk for the predicate 'role' is $SC_Q^{role} = \langle role, SC_Q^{employee\_name} \rangle$.

Unknown Predicate as wh-word. In a domain 'who' usually refers to a person, such as 'employee_name', 'student_name'; 'when' refers to date/time attributes like 'joining_date', 'completion_time'; and 'where' refers to locations like 'address', 'city'. For the given business application, this information about the wh-words is identified, and stored in the seed ontology. In questions involving any of these wh-word, the predicate corresponding to the wh-word is found using the domain ontology, which might be a possible candidate for being a unknown predicate. If the wh-word in the question is compatible to more than one predicate in the domain, then more semantic chunks - corresponding to each compatible predicate - are obtained. Semantic information is used in such cases to resolve the ambiguity regarding the most appropriate predicate (See Section 6.2). The constraints of the wh-word are determined on the basis of the role of the wh-word in the question as discussed below.

- If the wh-word is the subject in the question, the corresponding verb phrase determines the constraint on the wh-word. For example, consider the query 'Who is the project leader of Bechtel?' In this question, the verb phrase 'is the project leader of Bechtel' gives the constraints of 'who'. The constraints embedded in this verb phrase are 'role=project leader' and 'project_name=Bechtel'. Since in our domain 'who' corresponds only to the predicate 'employee_name', the semantic chunk is $\langle employee\_name,\ role = project\ leader,\ project\_name = Bechtel \rangle$.
- In other cases, the words in the phrase enclosing the wh-word determines the constraints on the wh-word. For example, for the question 'When did Ritesh join Bechtel?', the chunk structure as given by a NL parser is '[S When did [NP Ritesh NP] [VP join [NP Bechtel NP] VP] S]'. In our domain 'when' corresponds to 'joining_date'. The constraints in this question are 'employee_name=Ritesh' and 'project_name=Bechtel'. Thus, the semantic chunk obtained is $\langle joining\_date,\ employee\_name = Ritesh,\ project\_name = Bechtel \rangle$.

Wh-word as the Determiner of the Unknown Predicate. In this case also the constraints are determined as discussed above. For example, consider the question 'In which project is Ritesh allocated?'. The constraint for the unknown predicate 'project_name' can be identified as 'employee_name=Ritesh'. Thus the semantic chunk is $\langle project\_name,\ employee\_name = Ritesh \rangle$.

Using the syntactic information as discussed above, all possible semantic chunks for a parse structure of the question are determined. The set of these chunks is a semantically viable chunk set only if the chunk set satisfies the conditions (a) and (b) specified in the definition of SVC sets. For instance, for the query in Example 1, two SVC sets are obtained as given in Figure 2.

$SVC_{Q_1} = \{SC_{Q_1}^{role}, SC_{Q_1}^{employee\_name}, SC_{Q_1}^{project\_name}\}$, where:
- $SC_{Q_1}^{project\_name}$ = (*project_name*, *project_type* = *SWON*, *costing* > $15000, *revenue* < $25000);
- $SC_{Q_1}^{employee\_name}$ = (*employee_name*, *joining_date* < 18/10/2008);
- $SC_{Q_1}^{role}$ = (*role*, $SC_{Q_1}^{project\_name}$, $SC_{Q_1}^{employee\_name}$).

And,
$SVC_{Q_2} = \{SC_{Q_2}^{role}, SC_{Q_2}^{employee\_name}, SC_{Q_2}^{project\_name}\}$, where:
- $SC_{Q_2}^{project\_name}$ = (*project_name*, *project_type* = *SWON*);
- $SC_{Q_2}^{employee\_name}$ = (*employee_name*, *joining_date* < 18/10/2008, *costing* > $15000, *revenue* < $25000);
- $SC_{Q_2}^{role}$ = (*role*, $SC_{Q_2}^{project\_name}$, $SC_{Q_2}^{employee\_name}$).

Fig. 2. SVC Sets for Example 1

If for a query Q, only one semantically viable chunk set is found then this chunk set is the semantically valid chunk set. In other cases, the semantically valid chunk set is found by using the domain specific semantic information as discussed in the following section.

6.2 Finding Semantically Valid Chunk Sets

If more than one semantically viable chunk sets are obtained for a question, semantic information obtained from the domain ontology is used to determine the semantically valid chunk set. Let $SVC_{Q_1} = \{SC_{Q_1}^{p} | p \in P_Q^U\}$ and $SVC_{Q_2} = \{SC_{Q_2}^{p} | p \in P_Q^U\}$ be any two SVC sets for a query Q. Since there are more than one SVC set for Q, $\exists p_i, p_j \in P_Q^U$, and $c' = (p', v') \in C_Q$ such that c' is a constituent of $SC_{Q_1}^{p_i} \in SVC_{Q_1}$ and $SC_{Q_2}^{p_j} \in SVC_{Q_2}$. But, in the valid interpretation of Q, c' can specify either the unknown predicate p_i or the unknown predicate p_j. Hence we conclude that, in this case, the syntactic information is not sufficient to resolve the ambiguity whether c' is a constraint of p_i or p_j.

For instance, consider the SVC sets of the query in Example 1 (Figure 2). In the two SVC sets obtained for this query, the constraints '`costing > $15000`' and '`revenue < $25000`' are bound to '`project_name`' in SVC_{Q_1}, and to '`employee_name`' in SVC_{Q_2}.

To resolve such ambiguities, we use *depth* between the concerned predicates. The number of tables required to be traversed[4] in order to find relationship between any two predicates is determined through the domain ontology. This is referred to as the depth between the two predicates. If for a pair of predicates,

[4] This is found using the primary and foreign key information of the tables.

there exists more than one path then we choose the one with the minimum depth. It is observed that the semantic chunk in which the unknown predicate and the constraint predicate pair has lesser depth is the one which is more likely to be the correct pair. We use domain ontology to find the depth between two predicates as described below.

Step 1. Breadth first search (BFS): The system does a BFS on the tables in the ontology to determine if p_i or p_j belongs in the same table as that of p'. Without loss of generality, assume that p_i and p' belong to the same table, and p_j does not belong to the table of p'. In this case, $SC_{Q_1}^{p_i}$, and consequently SVC_{Q_1} is assumed to be correct, and SVC_{Q_2} is rejected. This in this case, $SVaC_Q = SVC_{Q_1}$.

Step 2. Depth first search (DFS): We involve DFS method to resolve the ambiguity regarding the constraint c' if BFS is not able to do so. The depth of the path from p' to p_i and p_j is found using the domain ontology. The constraint c' is attached to the predicate with which the distance of p' is minimum, and the corresponding SVC set is the semantically viable chunk set.

In Example 1 (Section 4), the constraint predicates '`revenue`' and '`costing`' are found to be closer to the predicate '`project_name`' than to the predicate '`employee_name`'. Hence, the semantically viable chunk set SVC_{Q_1} is the semantically valid chunk set.

An advantage of this approach is that depending upon the question complexity the system does a deeper analysis. Domain ontology is used only if a question cannot be resolved by using just the syntactic information. If domain information also is not sufficient for question interpretation, then answers for all interpretations are found, and the user is asked to choose the correct answer.

7 Formal Query Formation

The semantic chunks of the SVaC set are processed by the QueryManager. In this module, a formal query is generated on-the-fly from the semantic chunks to extract the answer of the user's question. Since the domain ontology is in RDF format, we generate queries in SPARQL[5] which is a query language for RDF.

For a semantically valid chunk set, we start with formulating SPARQL queries for the semantic chunks which do not contain any sub-chunk. The unknown predicate of the semantic chunk forms the 'SELECT' clause, and the constraints form a part of the 'WHERE' clause.

For instance, from the SVaC set $SVaC_Q$ (i.e., SVC_{Q_1} in Figure 2) of Example 1, we first formulate SPARQL query for the semantic chunks $SC_{Q_1}^{project\_name}$ and $SC_{Q_1}^{employee\_name}$, and obtain the answer from the RDF. Assume that the answers of these chunks are obtained as "[`Quantas`, `Bechtel`, `NYK`]", and "[`Rajat`, `Nidhi`]". The answers obtained from independent semantic chunks are then substituted in the semantic chunks involving nested sub-chunks. For example, to formulate the query for the semantic chunk $SC_{Q_1}^{role}$, the answers of the semantic

[5] http://dev.w3.org/cvsweb/2004/PythonLib-IH/Doc/sparqlDesc.html?rev=1.11

chunks $SC_{Q_1}^{project\_name}$ and $SC_{Q_1}^{employee\_name}$ are used. Therefore, upon substituting these answers, the semantic chunk $SC_{Q_1}^{role}$ is modified as:
$SC_{Q_1}^{role} = \{role, project\_name = Quantas, project\_name = Bechtel, project\_name = NYK, employee\_name = Rajat, employee\_name = Nidhi\}$. The SPARQL query for this chunk is then generated, and answer of the user query is retrieved.

8 Experimental Results

To test the approach discussed above, we carried out experiments on various domains (project management, retail, asset management). Users were asked to pose queries to an existing question answering system [8]. A set of almost 2750 questions were posed to the system, out of which approximately 35% consisted of fairly simple questions (No chunking required, but predicate-value binding required) e.g. "`List all projects with costing less than $30000`". The remaining 65% questions required correct predicate-value binding as well as correct chunking, like "`list the employees in the projects having costing less than $30000, started on 2009-10-10 with Ritesh as group leader`". When compared with actual answers following observations were made:

- 791 out of 946 questions were answered correctly for simple questions, which sums up approximately to 83.6%.
- For complex queries, 431 out of 1804 were correctly answered which approximately accounts 23.9%

The users' questions were again tested on the new system as discussed in this work. We have used Link Parser [10] for the syntactic analysis of the queries. The link parser yields the syntactic relations between different words of the sentence in the form of labeled links. In case of ambiguity in the input sentence, the link parser yields syntactic structures corresponding to all possible interpretations.

The following observations were made:

- 863 out of 946 questions were answered correctly for simple questions, which sums up approximately to 91.2%.
- 1508 out of 1804 were correct from complex questions that accounts 83.6%

Comparing the results of the two approaches, a direct increase of about 7% was attained for simple questions and for complex queries, it went up by almost 58%. The increment in the correctness of the answers were due to the following:

- Due to the predicate object binding, correct bindings were obtained (even for simple questions) which increased the number of answers correctly.
- The chunks obtained from the link parser when furnished with domain knowledge in the form of constraints, helped in achieving higher correct results.

Approximately 13% of the questions were not answered at all, or were answered incorrectly, or in some cases partially correct answers were obtained. After the analysis as to why the answers to those queries were not obtained, following conclusions were made.

- Syntactic analysis by the parser was not correct.
- The question posed by the user was grammatically incorrect.
- The system was not able to capture the semantic knowledge. E.g. for the queries '`Who is reporting to Ritesh?`' and '`Who Ritesh is reporting to?`", the system could not fetch correct answers.

We are currently working towards handling these issues.

9 Conclusion

We have described an approach to obtain semantically valid chunk set for NL-enabled business applications. For any question posed by a user to a business application system in natural language, the system should be robust enough to analyze, understand and comprehend the question and come up with the appropriate answer. This requires correct parsing, chunking, constraints formulation and sub-query generation. Although most general purpose parsers parse the query correctly, due to lack of domain knowledge, domain relevant chunks are not obtained. Therefore, in our work we have concentrated on enriching general purpose parsers with domain knowledge using domain ontology in the form of RDF. We have handled constraints formulation and sub query generation which form the backbone of any robust NL system. Tackling all these issues make any natural language-enabled business application system more robust, and enables it to handle even complex queries easily, efficiently and effectively.

References

1. Lopez, V., Motta, E., Uren, V., Sabou, M.: State of the art on semantic question answering - a literature review. Technical report, KMI (May 2007)
2. Androutsopoulos, I., Ritchie, G., Thanisch, P.: Natural language interfaces to databases - an introduction. Natural Language Engineering 1(1) (1995) 29–81
3. Antoniou, G., van Harmelen, F.: A Semantic Web Primer. The MIT Press (2004)
4. Popescu, A.M., Armanasu, A., Etzioni, O., Ko, D., Yates, A.: Modern natural language interfaces to databases: Composing statistical parsing with semantic tractability. In: 20th international conference on Computational Linguistics, Geneva, Switzerland (2004)
5. Charniak, E.: A maximum-entropy-inspired parser. In: NAACL. (2000)
6. Katz, B., Borchardt, G., Felshinm, S.: Syntactic and semantic decomposition strategies for question answering from multiple resources. In: AAAI 2005 Workshop on Inference for Textual Question Answering, Pittsburgh, PA (July 2005) 35–41
7. Lopez, V., Motta, E., Uren, V.: Aqualog: an ontology-driven question answering system to interface the semantic web. In: NAACL on Human Language Technology, New York (2006) 269 – 272
8. Bhat, S., Anantaram, C., Jain, H.K.: A framework for intelligent conversational email interface to business applications. In: ICCIT, Korea (2007)
9. Klein, D., Manning, C.D.: Fast exact inference with a factored model for natural language parsing. In: Advances in Neural Information Processing Systems. Volume 15., Cambridge, MA, MIT Press (2003) 3–10
10. Grinberg, D., Lafferty, J., Sleator, D.: A robust parsing algorithm for link grammars. In: Fourth International Workshop on Parsing Technologies, Prague (September 1995)