QUASL: A Framework for Question Answering and its Application to Business Intelligence

Nicolas Kuchmann-Beauger, Falk Brauer, Marie-Aude Aufaure

QUASL: A Framework for Question Answering and its Application to Business Intelligence

Nicolas Kuchmann-Beauger
SAP France
Levallois-Perret, France
nicolas.kuchmann-beauger@graduates.centraliens.net

Falk Brauer
SAP Asia Pte Ltd
Singapore
falk.brauer@sap.com

Marie-Aude Aufaure
École Centrale Paris
Châtenay-Malabry, France
marie-aude.aufaure@ecp.fr

***Abstract*—Question Answering (Q&A) from structured data is a technique that may revolutionize enterprise search. A very promising use-case for such technology is Business Intelligence (BI). In order to make BI more accessible to end-users, some efforts have been made in the field of search for existing reports. However, the problem of converting an end-user's natural language input to a valid structured query in an ad-hoc fashion hasn't been sufficiently solved yet. In this paper we present a framework for Q&A systems that operate on structured data. The main innovation is that the framework allows defining a mapping between recognized semantics of a user's questions to a structured query model that can be executed on arbitrary data sources. It bases on popular standards like RDF and SparQL and is therefore very easy to adapt to other domains or use-cases. We will describe the application of this framework at hand of a BI question answering use-case, which also includes the personalization of generated queries, demonstrating the real-world applicability of our approach. In our experiments, we demonstrate that with our approach one can easily achieve a similar answering quality as one of the most popular Q&A systems on the Web.**

I. Introduction

In the last decades data warehouses became an important information source for decision making and controlling. A lot of progress has been made to support casual end-users by allowing interactive navigation inside complex reports or dashboards (e.g. by interactive filtering or calling OLAP-operations such as drill-down in a user-friendly way). In addition there has been a lot of effort in making reports or dashboards searchable. However, most casual users still have to rely on pre-canned reports that are provided by the IT-department of a company because todays' Business Intelligence (BI) self-service tools still require a lot of technical insights such as an understanding of the data warehouse schema. This is especially cumbersome because data warehouses grew dramatically in size and complexity. A popular use-case for BI is for instance the segmentation of customers to plan marketing campaigns (e.g. to derive the most valuable, middle-aged customers in a certain region). It is not unusual that business users who plan a campaign have to cope with hundreds of key performance indicators (KPIs) and attributes, which they have to combine in an ad-hoc fashion to cluster their customer base. A keyword or even natural language-based interface to formulate their information need would ease this task a lot. This can be underlined with the recent success of question answering systems, such as WolframAlpha[1], especially in conjunction with speech-to-text technologies like Siri[2] and the huge efforts in the database community to enable keyword-based search in databases (e.g. [1], [2], [3]).

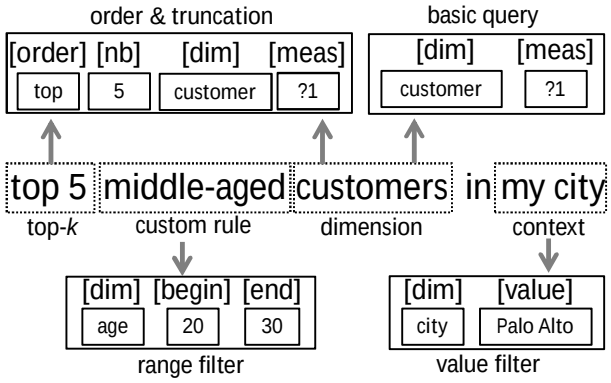
However, the keyword based approaches developed so far lack many important features to fully enable a question-driven data exploration by end-users, where the consideration of range queries, the support to include application-specific vocabulary (e.g. "middle-aged") or leveraging of the users' context (e.g. "customers in my region") are only the most obvious ones. Note that the problem is not only to extract semantics from a user's question (e.g. from a range phrase such as "between 1999 and 2012"), which is supported by our framework as well. The more important problem is to relate findings detected in a user's question to formulate a well-defined structured query. The framework presented in this paper supports the whole process of defining and executing a domain or application-specific Question Answering system. We provide a basic infrastructure for entity recognition, which we briefly introduce in section IV. The main innovation, i.e. the declarative description of constraints on the user input and background knowledge; definition of variables to be used in the structured query and the mapping of this variables into arbitrary structured queries will be discussed in section V and section VI. In section VII, we elaborate the ranking of computed structured queries. The experimental evaluation of our framework can be found in section VIII and the related work in section IX. However, before going into the details of our approach, we like to introduce the problem in section II and give an overview on the system and the data structures that we leverage in section III.

II. Problem statement

In general, the problem of Question Answering (Q&A) from structured data can be formalized as follows: given a structured query language L and a user's question q, we define a mapping $q \mapsto R$ to a ranked list R (results) of structured queries $r_1, r_2, \ldots, r_n \in L$, where r_i represents the i-th highest scored interpretation of the user's questions with respect to the data and metadata of the underlying structured data source. We focus in this paper on multi-dimensional queries (structured queries) and data warehouses (data sources), without restricting the generality of our approach. A simple multi-dimensional query (see [4] for a more detailed definition) is usually represented by a number of dimensions (organized in hierarchies) or their attributes, measures (aggregated KPIs with respect to the

[1] See http://www.wolfram.com/mathematica/.

[2] http://www.apple.com/iphone/features/#siri.



(a) A user's question and derived semantic units (comparable to a parse tree in natural language processing). Successive tokens that satisfy some constraints (e.g. linguistics pattern or dictionary matches) are marked with dotted boxes. Inferred semantics are drawn with solid rectangles. These semantic units of recognized entities form parts of potential structured queries that might fullfill the users information need. In addition, the system has to propose a measure for '?1' to compute a valid multi-dimensional query.

```
SELECT
 sum(Invoice_Line."DAYS"
  * Invoice_Line."NB_GUESTS"
  * Service."PRICE") AS revenue,
 Customer."LAST_NAME" AS customer
FROM City
INNER JOIN Customer
 ON (City."CITY_ID"=Customer."CITY_ID")
INNER JOIN Sales
 ON (Sales."CUST_ID"=Customer."CUST_ID")
INNER JOIN Invoice_Line
 ON (Invoice_Line."INV_ID"=Sales."INV_ID")
INNER JOIN Service
 ON (Invoice_Line."SERVICE_ID"=Service."SERVICE_ID")
WHERE
 city = 'Palo Alto' AND
 age >= 20 AND
 age <= 30
GROUP BY
 customer
ORDER BY revenue
LIMIT 5
```

(b) Example SQL query that was generated from the user's question in figure 1a. Natural language patterns, constraints given by the data and metadata of the data warehouse (see figure 2) have been applied to infer query semantics. This was mapped to a logical, multi-dimensional query, which in turn was translated to SQL. Note, that the 'revenue' represents a proposed measure, depicted as '?1' in figure 1a. The computation of the measure 'revenue' and the join paths are configured in the metadata of the warehouse.

Fig. 1: Translating a user's question into a structured query.

used dimensions or attributes) and filters. It is executed on top of a physical or virtual data cube (through an abtraction layer generating a mapping to SQL in our case). In addition, result modifiers (e.g. for sorting and truncation) can be added to a query. The interested reader might already look up the logical schema of the example data warehouse used throughout this paper in figure 2. An example question, q="Top 5 middle-aged customers in my city", is shown in figure 1a, while listing 1b depicts an example result query $r \in R$ in SQL syntax. Note that the join paths in the example, the aggregation function `sum()`, and the expression inside the aggregation function representing the measure 'revenue' are predefined in the data warehouse metadata. We also depict in figure 1a intermediate steps to derive the final structured query. Successive tokens that satisfy some constraints (e.g. 'customers', which matches the name of an entity in the data warehouse metadata) are marked with dotted-line boxes. Inferred semantics (e.g. the filter for the age range) are drawn with solid-line rectangles.

In order to derive a structured query $r \in R$ from a given question q, we have to solve a series of problems, which form a kind of process and which we detail in the following:

(1) Information Extraction: The first step in the processing is to derive lower-level semantics from the user's question. This is, data and metadata of the underlying structured data source (i.e. domain terminology) have to be recognized (e.g. 'customers' refers to the dimension 'customer' in the data warehouse schema). To support more sophisticated queries (e.g. range queries) and go beyond keyword-like questions, we allow administrators to define custom vocabulary (such as 'middle-aged') and more complex linguistic patterns, which simple example is 'top 5'. Such artifacts may export variables, such as the beginning and ending of the age range for 'middle-aged' or the expected number of objects for 'top 5'.

(2) Normalization and Transformation: In many cases, we cannot map the users' input directly to a structured query. For instance dates or numbers may be expressed in different forms (e.g. '15000' as '15,000', '15k' or even 'fifteen thousand'). The same holds true for custom vocabulary such as 'middle-aged', which translates to a range on the 'age' attribute of the dimension 'customer' (see figure 2). Therefore, we need a configurable mechanism to translate input variables derived from linguistic patterns to variables that conform with the query language L. Similarly, we require a mechanism to define variables for custom vocabulary (e.g. 'middle-aged' defines a range starting at '20' and ending at '30').

(3) Structural Constraints and Missing Artifacts: One of the most challenging problems is to generate valid structured queries (valid in the sense that $q \in L$ holds and that q returns a – maybe empty – result). To go deeper into our example, a series of constraints need to be considered to generate valid multi-dimensional queries in a next processing step, such as:

- Artifacts used within a query have to occur in the same data warehouse (if several ones are used).
- A query contains at least one measure or dimension.
- A query that contains two dimensions requires one or more measures that connect these dimensions (see figure 2), which implies that these dimensions relate to the same fact table.
- Different interpretations for ambiguously recognized dimensions (i.e. different matches covering the same text fragment in the user's question) shall not be used within the same structured query because it would change the aggregation level.
- A sorting or truncation criteria (e.g. 'top 5') requires usually an assigned measure (note that several sortings and truncations can be applied within one query, e.g. for a question like 'Revenue for the top 5 cities and the top 10 resorts'). If no measure for a sorting or truncation criteria can be extracted from the question, the system has to propose at least one

measure, e.g. ‘revenue’, as shown in listing 1b for the missing input ‘*?1*’ in figure 1a.

Other constraints that occur in the BI domain are imposed by the specific application and consideration with respect to the users’ expectation. Our example application aims to provide insightful vizualisations to the user’s business. Thus, a user who types ‘revenue’ as question is not necessarily interested in the total revenue achieved by the company since its existence (which a query consisting only of the measure would return). Most likely, he is interested in a specific aspect such as the ‘revenue per year’ or ‘revenue per resort’, even though he is not yet sure, which of these aspects he would like to explore. Therefore, our application proposes in such situations dimensions that form together with the measures of the user’s question a valid structured query. Same holds true if the user does not mention a measure, but only a dimension as shown in the example in figure 1a. In this case, the system proposes related measures (here ‘revenue’). Another interesting aspect is contextualization. A user might have a user profile, which augments the systems metadata (e.g. with the city the user is located in). In order to simplify the data exploration and return more relevant data for his current task, we may leverage the user profile to impose additional filters. In figure 1a the user asks for instance for ‘my city’. Knowing that there is a mapping between ‘Palo Alto’ in the user profile and a value inside the data underlying the dimension ‘City’, we may automatically generate a strucured query with a filter for ‘Palo Alto’. Considering all these structural constraints and possibilities to augment queries with artifacts from the underlying metadata, it is obvious that such constraints are difficult to express and maintain in an imperative programming model. In section V we will present as one of our major contributions a declarative way to capture such domain or application-specific constraints to generate structured queries.

(4) Mapping Artifacts into Structured Queries: Once we have defined constraints and artifacts that shall be considered, such as dimensions, measures, attributes and custom variables (e.g. for ‘top 5’ or ‘middle-aged’), they have to be mapped into semantic units (e.g. ‘basic query’ or ‘range filter’ as shown in figure 1a). Now we have to create a new data structure that represents a query such that artifacts refered in the constraints are mapped to the respective parts of a structured query. These are usually *projections* (i.e. measures and dimensions), *selections* (i.e. filters) and *result modifiers* (such as ordering or truncation expressions). Again, different query languages, domains and applications may impose different requirements. Clearly, measures and dimensions that where recognized as ‘basic query’ in figure 1a have to be included in the projections. ‘Age’ as dimension refered by the range filter derived from ‘middle-aged’ shall not be included in the projections, since it would change the semantic of the query if included in the `GROUP BY`-statement in listing 1b. In general, it is often application-specific whether a dimension used within a filter expression shall be mapped to a projection. In our case, there is a direct mapping from the query result to a chart. Since most traditional chart types (e.g. bar charts) support only 2 dimensions, we have to reduce the number of projections to a minimum and therefore split very complex queries into several ones, each showing different views on the data set. Another way of reducing the number of projections is to neglect projections for filters having only one value (e.g. as shown for ‘Palo Alto’ in figure 1b) since it does not change the semantics of the query.

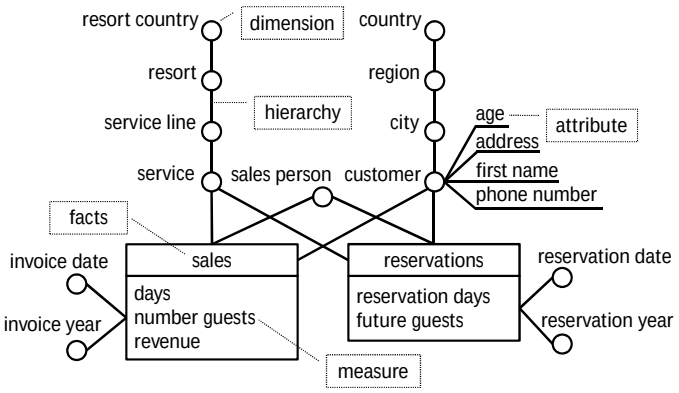


Fig. 2: Logical data model of an example data warehouse in notation proposed by Golfarelli et Rizzi [5]: two fact tables (reservation, sales) define different measures (e.g. revenue or reservation days). These facts connect the different dimensions (e.g. customer and sales person). Dimensions are organized in hierarchies (e.g. customer, city, region, etc.) and may have assigned a set of attributes (e.g. age for customer).

(5) Scoring of Queries: The previous step generates potentially several structured queries which then have to be ranked by relevance with respect to the user’s question. We propose a series of heuristics for ranking generated queries in section VII.

Finally, ranked queries have to be executed. In our approach, we generate a intermediate query model that is than serialized and executed on the data source.

III. Metadata Management

An important foundation of the overall framework is the metadata management and the runtime information captured in the so called *parse graph*. We use the term *parse graph* to state its close relationship to the term *parse tree*, often used in the context of natural language processing. In our case the *parse graph* and other metadata required to interpret a question is captured in form of RDF[3] since it is a widely-accepted standard for representing graphs. We also benefit a lot from the power of the graph pattern query language SparQL[4] as detailed later on. Note that we use in the remaining paper the terms *resource* when we refer to the actual RDF-representation and *node* when we describe higher level concepts (even though they can be seen as synonymous in this paper). In figure 3 we show an example of a parse graph for our example from figure 1 and other graph-organized metadata. Before discussing the parse graph itself, we like to detail the metadata graphs.

On top in figure 3 we see a graph capturing the user profile and below an excerpt of the graph representing the data warehouse’s schema. We only show one data warehouse (see ‘Resorts’ node) for brevity and only one measure (‘Revenue’), two dimensions (‘Customer’ and ‘City’) and one attribute (‘Age’). The node ‘City’ is linked to the location node (‘Palo Alto’) from the user profile. Currently this link is automatically

[3] see http://www.w3.org/RDF/.

[4] SPARQL Protocol and RDF Query Language

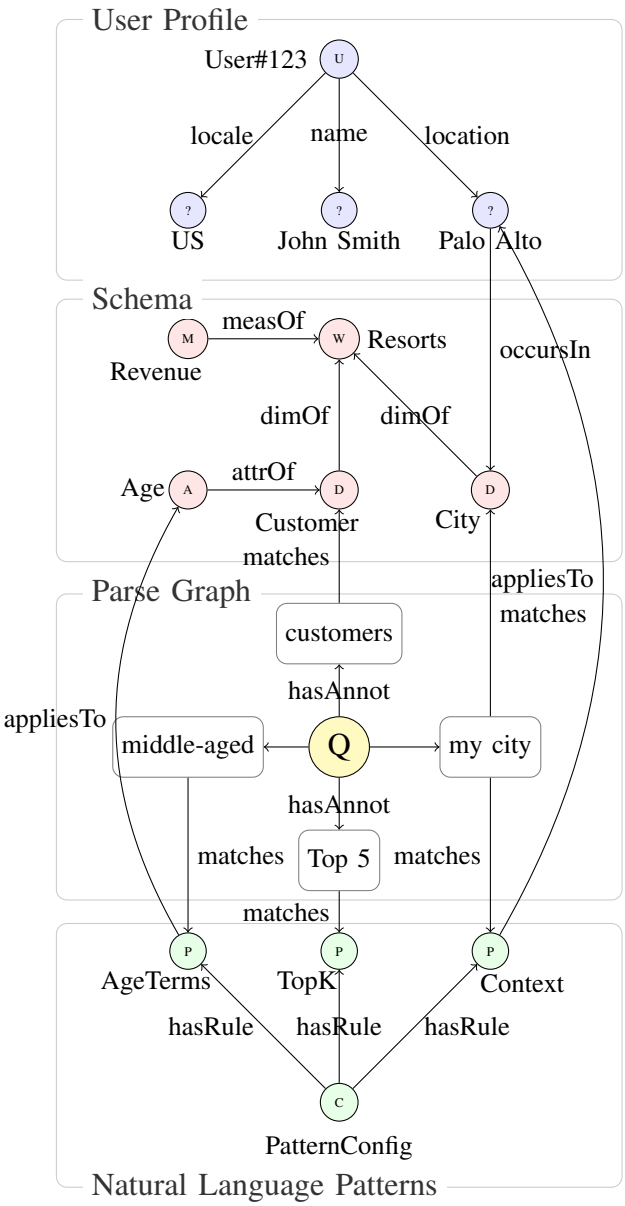


Fig. 3: Used metadata and parse graph of an example question

established by matching the values of the user profile against the warehouse's data. On the bottom we see a graph capturing metadata of some configured natural language patterns (see next section). We keep this information inside RDF to relate these patterns to other resources. For instance the custom vocabulary for the terms related to age (node labeled with 'AgeTerms') – used to identify 'middle-aged' – applies to the schema's attribute 'Age' by its pattern definition. The user context pattern ('Context'-node) relates to all user profile nodes that have a corresponding value in the warehouse's data (here: 'Palo Alto'). In addition, the nodes of the natural language pattern graph have properties like an executable information extraction rule (see section IV) and a set of variables that can be exported, e.g. that the 'TopK'-pattern exports the number of items and that the ordering is ascending (cf. figure 1a).

The parse graph itself is depicted in the third box from the top. It is generated by Information Extraction algorithms. The graph mainly consists of a central node representing the user's question (the larger node marked with 'Q') and so-called annotation nodes (depicted with a rectangle shape in figure 3, labeled with the corresponding fragment of the users' question). Annotation nodes capture metadata that was aquired during the matching process (e.g., when matching data warehouse's schema or natural language patterns) and link to relevant resources used or identified during this process (e.g., a dimension or the natural language pattern that were used). As runtime metadata we keep for instance the position of a match (offset and length of the matched fragment within the question), the type of the occured match (e.g., match in data warehouse metadata or match with natural lanugage patterns) and a confidence value. In addition Information Extraction algorithms may capture specific metadata such as instantiated output variables for natural language patterns (e.g., the '5' extracted from 'Top 5'). Before going into the details on how to translate semantics captured during the information extraction process into structured queries, we like to detail how to derive a structure such as the one shown in figure 3 and how to configure natural language patterns.

IV. Entity recognition

To derive a graph structure such as shown in figure 3 we need to match the user's question with the metadata and data of the underlying structured data source and apply all configured natural language patterns. In the beginning of this section, we briefly introduce the matching algorithms.The last subsection contains a more detailed description on how natural language patterns are configured and executed.

A. Matching Metadata and Data

We use a state-of-the-art information extraction system (SAP BusinessObjects Text Analysis™, a succesor of the system presented in [6]) with a custom scoring function to match metadata objects inside the user's question. As scoring function for evaluating individual matches we adapted the scoring that we presented in [7]. In a nutshell, it combines TF-IDF like metrics with Levenshtein and punishes in addition matches where the length of a term in the metadata is much longer than the string that occurs in the users' question. A threshold on the score limits the number of matches that are considered for further processing.

If a substring of the users' question was identified as matching a term in the datasource's metadata (user profile or schema), the component generates an annotation node in the parse graph. This node links the matched node and the question node (cf. figure 3). As discussed before, runtime metadata such as the offset, length and score of the match are stored as attributes of the annotation node. Matching dimension values (used for filters) works in a similar fashion. We did some optimizations by leveraging the full-text search capabilities of the underlying database. If a match with some dimension value (e.g. 'Palo Alto') occurs, the system creates an annotation like for matched metadata, linking the question and the metadata node, to signal the system that a value for a dimension was identified. The difference between metadata annotations mentioned above to the ones created for dimension values is the annotation type which is assigned to the annotation and the relation to the metadata node (i.e. 'hasValue' instead of 'matches').

```
1 :yearsBackWardsToDateRange
2   rdf:type features:NlpPattern;
3   rdfs:label "Computes the beginning and end date, given a time range in years"^^xsd:string;
4   nlp:outputVariables "yearsBack,rangeBegin,rangeEnd"^^xsd:string;
5   nlp:rule
6     "((<last>|..|<previous>)([OD yearsBack]<POS:Num>[/OD])?<STEM:year>)"^^xsd:string;
7   nlp:computeVariablesWithScript """
8     var today = new Date(); var dd = today.getDate(); var yyyy = today.getFullYear();
9     rangeEnd =  calculateEnd(dd,mm,yyyy);
10    rangeBegin = calculateStart(dd,mm,yyyy,yearsBack);
11
12    function calculateEnd(dd,mm,yyyy) {
13        return  yyyy + '-' + mm + '-' + dd;
14    }
15    function calculateStart(dd,mm,yyyy,yearsBack) {
16        return (yyyy-yearsBack) + '-01-01';
17    } """^^xsd:string ;
18  nlp:appliesTo dataType:Date;
```

Listing 1: Pattern to compute dates from 'last 3 years'.

B. Natural Language Patterns

We developed a very powerful mechanism for natural language pattern, which we like to introduce in more detail. It can be used to implement custom functionality (e.g. range queries, top-k queries or custom vocabulary such as shown for "middle-agged" in figure 1a) that goes beyond keyword-matching. As explained in the previous section, natural language patterns are configured using RDF (see listing 1 for an example). The three main parts of a natural language patterns are:

(1) Extraction Rules: The basis for natural language patterns are extraction rules. In our case we use the CGUL rule language[5], which can be executed using SAP BusinessObjects Text Analysis™. It bases similarly as CPSL [8] or JAPE [9] on the idea of *cascading finite-state grammars* meaning that extraction rules can be built in a cascading way. Thus any other rule engine can be used for this purpose. We make heavy use of built-in primitives for part-of-speech tagging, regular expressions and the option to define and export variables (e.g. the '5' in 'top 5'). Note, that a rule might simply consist of a token or a phrase list, e.g. containing 'middle-aged'.

(2) Transformation Scripts: Once a rule fired, exported variables may require some post-processing , e.g. to transform '15,000' or '15k' into '15000', a expression that can be used within a structured query. In many cases there is also the need to compute additional variables. The most simple case for such functionality is to output beginning and ending of the age range defined by a term such as 'middle-agged'. To do additional computations and transformations, we allow to embed scripts inside a natural language pattern, which can consume output variables of the extraction rule and can define new variables as needed.

(3) Referenced Resources: A rule is often specific for a resource in some metadata graph. For instance in figure 3 the pattern for 'AgeTerms' applies only to the dimension 'Age', the 'Context' pattern only to nodes within the user profile and other patterns apply only to certain data types (e.g. patterns for ranges to numerical dimension values) – which are also represented as nodes. In order to restrict the domain of patterns, we allow to specify referenced resources. Later, we will detail how these references can be used in generating structured queries.

We can see an example natural language pattern in listing 1. It does not match our running example question to underline the power and flexibility of the described mechanism. It depicts a pattern to compute from a phrase like 'for the last 3 years' two date expressions (namely beginning and ending date) that can be used in a structured query. The example pattern is presented in the Turtle RDF format[6].

The first line defines the URI of the pattern (i.e. the subject of all following properties). All remaining lines define the pattern's properties (in terms of predicates and objects). Line 2 and 3 contain the type and description in the sense of RDF and RDF-Schema. In line 4 we define the variables that are output of the pattern, here 'yearsBack' and the actual dates ('rangeBegin' and 'rangeEnd').

The extraction rule is defined in line 5 and 6. It consists of some trigger words like 'last' or 'previous', the exported number (`[OD]` marks that the expression between shall be exported, `<POS:Num>` references the part-of-speech tag for numbers) and the ending token 'year' (and its stems).

Between line 7 and line 17 stands the script used to compute the actual values for the variables 'rangeBegin' and 'rangeEnd'. We use JavaScript, because it can be executed easily in our host programming language (Java) and embed it into the RDF representation to store the rule definition and the transformation logic together. In the last line, we define that this rule only applies to dimensions which have values of data type 'Date'.

Once an extraction rule fired and the attached script has been evaluated, an annotation node in the parse graph is created as shown in figure 3. The annotation node cares as properties runtime metadata such as the match position (again offset and length inside the user's question), the annotation type and the computed variables.

V. Structural constraints

Once the *parse graph* is created for a particular user's question, the system has to ensure domain- and aplication-specific constraints. In addition to constraints mentioned in

[5]http://help.sap.com/businessobject/product_guides/boexir4/en/sbo401_ds_tdp_ext_cust_en.pdf

[6]http://www.w3.org/TeamSubmission/turtle/

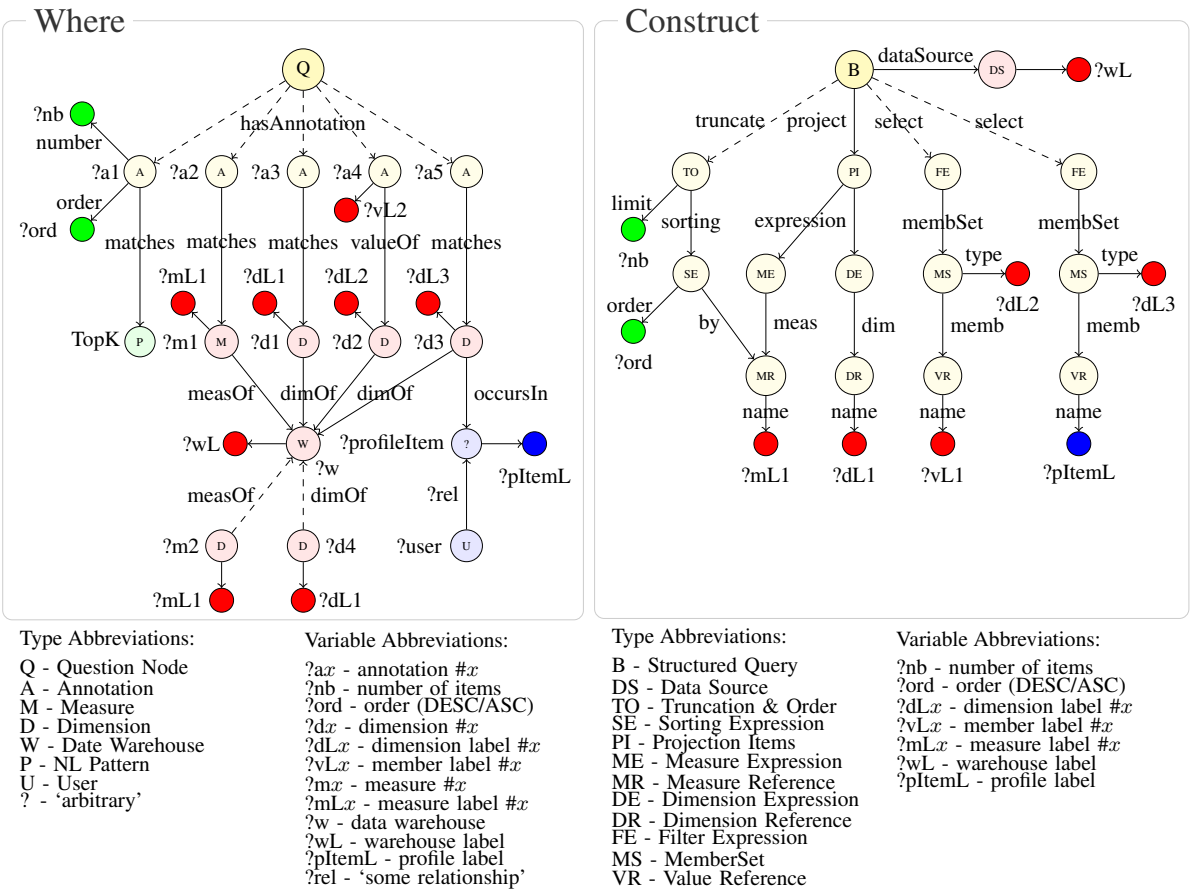


Fig. 4: Example for parse graph constraints and mapping rules to generate a structrued query (see figure 1)

section II for our BI use-case, other apply on how entities occur together in the query, the data types of the recognized entities or to which other entities they relate to. Another constraint takes the form of entity recommendation when the user did not include all necessary information to compute a valid query or to add additional filter for personalization. We discuss in the following two types of constraints: (1) *relational constraints* (see section V-A) which describe situations like the fact that a dimension and a measure should belong to the same data warehouse and (2) *property constraints*, which filter nodes based on property values (see section V-B), like the fact that two annotations are overlapping or close to each other. Then, we discuss a convenient feature of SPARQL 1.1[7] to inject additional variables (see section V-C), e.g. to to generate additional values to be used in the structured query if a certain graph pattern occurs.

A. Relational Constraints

SPARQL queries are essentially graph patterns. Nodes and edges are be adressed by URIs or are assigned to variables which are bound to URIs or literals by the SPARQL *query processor*. This mechanism of expressing graph constraints and of binding variables eases the configuration of our approach tremendously. Figure 4 is a vizualized example of complex constraints for selection and mapping rules that are used in our application setting. On the left-hand side stands an excerpt of the constraints and variables used in our BI use-case. The markers attached to a node represent in contrast to figure 3 assigned variables or URIs. URIs are expressed in short form and do not care a leading question mark. Edges between nodes and literals refer to `rdfs:label` if there are not marked otherwise. Dashed lines illustrate that a particular part of the graph pattern is optional (implemented trough an `OPTIONAL` statement in SPARQL). (Q) depicts the user's question. Below are annotation nodes and left to them assigned variable names (like '?a1') which form the *parse graph*. Other nodes reference metadata graphs (see figure 3). The nodes like (M) in figure 4 represent resources, while nodes like ● represent literals, which will be reused later for query composition. As discussed in next section, we map only literal variables to the final query model, to separate the input and output model on conceptual level. Note also that we define much more variables in the real-world use-case, e.g. to handover data types and scores from the question to the query model, which we leave out of the examples for sake of brevity. Our example exhibits constraints for the following situations:

Natural Language Patterns ('?a1'): The annotation ('?a1') addresses the natural language pattern for *TopK* (see figure 3). The *TopK* pattern exports the variables 'number' and 'order', which are bound to '?nb' and '?ord'. This rule might be combined with a rule triggered by phrases like 'order by ...' to assign a variable holding the dimension or measure such that an ordering can be applied. In general, different natural language patterns can be easily combined using *property*

[7] http://www.w3.org/TR/2012/WD-sparql11-query-20120724/

constraints as explained in next subsection. Patterns for ranges, custom vocabulary, *my*-questions etc. are treated similarly. In particular in situations related to ranges and the mapping of other data types, we define additional variables for the data type of certain objects (e.g. 'Date' or 'Numeric') to handle them separately in the final query generation. These attributes eventually influence the *serialization* of the structured query.

Data Warehouse Metadata ('?a2' and '?a3'): The annotations '?a2' and '?a3' refer to a recognized measure and dimension bound via a *matches* relationship in figure 4 and triggered by questions like "revenue per year". An arbitrary number of measures and two dimensions are allowed (due to the requirement of rendering charts). By assigning the nodes for the measure ('?m1') and dimension ('?d1') to the same node for the data warehouse ('?w') we ensure that these objects can be used together in a structured query and lead therefore to a valid query. More precisely we check whether recognized dimensions and measures are linked through a fact table (see figure 2). For reuse in the structured query, we assign the labels of the recognized objects to variables (i.e. '?mL1' for the measure label and '?dL1' for the dimension label). In some cases the system has to suggest fitting counterparts (e.g. *compatible* dimensions) to not aggregate all facts. In the example in figure 2 we choose '?d4' as dimension if the question contains only measures and '?m2' as measure if it contains only dimensions. Thus the system generates multiple interpretations for the user's question. The SPARQL blocks that contain '?d4' and '?m2' are optional and contain a filter (i.e. a *property constraint* as explained later) such that they are only triggered if either '?mL1' or 'dL1' are not bound. The label of the recommended measure or dimension is finally bound to the respective label variable that would otherwise be unbound (i.e. '?mL1' or 'dL1').

Data Warehouse Values ('?a4'): Instead of the *matches* relationship, we use the URI *valueOf* to assign the dimension value to the corresponding dimension (i.e. '?d2'). For later reuse, we assign the label of the value's node ('?vL2'), e.g. '2009' for a year, and the dimension value to a variable ('?dL2'). In the real-world use-case we consider not only one match situation (like in the example) but a couple of other situations, where the declarative approach is very valuable. For instance, we show here only the case where the matched value does not belong to an already-recognized dimension (i.e. 'd2' would be an additonal dimension in the query). For the situation where the value belongs to '?d1' – an already-recognized dimensions – we define another optional SPARQL block which is triggered by the *valueOf* relationship between the annotation and the corresponding dimension. We treat single value matches for one dimension differently than matches on multiple values that belong to the same dimension. Our declarative approach eases this, because another set of constraints can be simply defined with separate variables.

Personalization ('?a5'): Annotation '?a5' shows the personalization feature, which applies a filter for a dimension if a corresponding value is part of the user profile (see 'my city' in figure 1). The constraint captures following situation: an annotation ('?a5') refers to a dimension ('?d3') that *occursIn* some resource ('?profileItem') that has some relationship ('?rel') to the user ('?user'). From the graph pattern, we consider for later porcessing the label of the dimension ('?dL3') and the label of the user profile that occurs in this dimension ('?pItemL'). Note that constraints for personalization (as shown in figure 4) do not refer to the *my*-pattern (shown in figure 3) due to space constraints. If the constraints would be applied as shown here, we would simply test for every matched dimension whether there is a value mapping to the user profile. These examples highlight the flexibility of using SPARQL graph patterns to manage constraints and variables for query composition in Q&A systems. Addtitional constraints have to be applied on property or litteral level (see the following sub-section).

B. Property Constraints

The following details the use of constraints through SPARQL `FILTER` statements considered in addition to graph patterns. They are less important on conceptual level, but have many practical implications, e.g. to not generate duplicated queries or to add further functionality, which cannot be expressed on graph pattern level. The first obvious additional constraint is to check whether two annotations matching two distinct dimensions are different:

```
FILTER(!sameTerm(?a1, ?a2))
```

We mentioned in section II that it is often crucial to separate objects that matched the same part of the user's question into several structured query; this is even more important for dimension names because they define the aggregation level of the final result. This kind of constraints can be expressed using the metadata acquired during the matching process. Assuming that the position of a match inside the question has been assigned to the variable `?o1` and the offset and length of another annotation are assigned to `?o2` and `?l2`, the filter for ensuring that the latter one does not begin within the range of the first annotation can be expressed by:

```
FILTER(?o2 < ?o1 || ?o2 > (?o1 + ?l2)))
```

Property constraints are also used for more complicated query generation problems. For instance a generic *natural language pattern* for ranges would look similar as the one shown in figure 1. It would then apply to `dataType:Numeric`, be triggered by phrases like 'between x and y' and include a script for normalizing numbers. In combination with matched numeric dimension values, one can define a filter that tests whether two members where matched inside the matched range phrase and generate variables defining the beginning and ending of a structured range query.

C. Additional Variables

It is often usefull to define default values or to bind additional variables. An example for a default value would be to limit the size of the query results if it is not specified by the user. To do so, we add an optional SPARQL block that checks the variable '?nb' and assigns a value if it is unbound using: `BIND(1000 AS ?nb).`

There are plenty of other use-cases to inject additional variables, like defining analysis types (which are part of the not-illustrated metadata that is assigned to a structured query). These are indicators used to select the best fitting chart type for a single result. To capture the analysis type, we use certain trigger-words (see [10]) and additional constraints such as the number of measures and dimensions and the cardinality of dimensions. For instance we would select a *pie chart* if a single measure and dimensions is mentioned in the question and the user is interested in a 'comparison' (e.g. triggered by the term 'compare' or 'versus'). However, if the cardinality

of the dimension (which is maintained in the metadata graph) would exceed a certain value (e.g. 10), a *bar chart* would be a better fit because a pie chart would be difficult to interpret otherwise.

VI. Mapping to structured queries

For most of the cases like the example given in figure 1a, a structured query contains a *data source*, a set of *dimensions* and *measures*, a set of *filters* and an optional set of result modifiers, e.g. *ordering* or *truncation* expressions. For the example from figure 1a, a stuctured query could be represented as follows:

$$Q_1 = \left[\begin{array}{lcl} \text{data source} & = & \text{Resorts} \\ \text{dimensions} & = & \{\text{Customer}\} \\ \text{measures} & = & \{\text{Revenue}\} \\ \text{filters} & = & \left\{\begin{array}{lcl} \text{City} & = & \text{`Palo Alto'}, \\ \text{Age} & \geq & 20, \\ \text{Age} & \leq & 30 \end{array}\right\} \\ \text{truncation} & = & \{(\text{Revenue}, \downarrow, 5)\} \end{array}\right]$$

In there, curly brackets represent a set of objects, which might have a complex structure (e.g. for filters, which consist of a measure or dimension, an operator and a value). For truncations we use a triple consisting of the dimnesion or measure on which the ordering is applied, the ordering direction (ascending $\uparrow$, or descending $\downarrow$) and the number of items. Another intepretation for the user's question would be Q_2, which is similar to Q_1 except the proposed measure:

$$Q_2 = \left[\begin{array}{lcl} \ldots & & \\ \text{measures} & = & \{\text{Margin}\} \\ \ldots & & \\ \text{truncation} & = & \{\text{Margin}, \downarrow, 5)\} \end{array}\right]$$

Since the reprentation shown above captures only a fraction of the potential queries, we use RDF to capture the structure and semantics of the structured query which is than serialized to an executable query in a subsequent step. As discussed earlier, we define in the left part of figure 4 how to derive potential interpretations (i.e. variables and the constraints between them) using a SPARQL `WHERE` clause. Now we need to define the basic structure of a query (in RDF) and how to map variables into this model using a SPARQL `CONSTRUCT` clause (illustrated in the right part of figure 4). In this way, we separate the pattern matching, which can be quiet complex, from the actual mapping problem and ensure a fine-grained flexible control on how to generate structured queries.

Some of the most important concepts of our query model are illustrated in figure 4. On the top, stands the root node (B) defining a structured query. Below, dashed lines represent parts that are optional in the left side. These parts of the `CONSTRUCT` clause are only triggered if the respective variables are in the result of the `WHERE` clause, making it easy to describe alternative mappings for different situations as described in the *parse graph*. Besides of the actual query semantics, we attach some metadata nodes to the query node such as the *data source* (DS). It is bound to the variable '?w' representing the actual data warehouse upon which the generated query shall be executed. Additional nodes are dedicated to: *projection items* (PI), capturing all projections that are part of the final structured query; *filter items* (FI), expressing selections on a certain measure or dimension and *truncation and ordering* clauses (TO). The underlying structures are detailed in the following.

Projections The most important part of the actual query are projections, which in our use-case consists at least of one measure and dimension. To give a glimpse on our full query model and further detail the example, we define different kinds of expressions (via a common anchestor RDF type) where we depict here the subclasses *measure expression* (ME) and *dimensions expression* (DE). These nodes capture common metadata (not shown here), such as navigation paths (e.g. for drill-down operations) or confidence scores and refer to the actual object that defines the projection, here the *measure reference* (MR) and *dimension reference* (DR). They are in our case the labels of recognized objects. It does not matter whether we use the recognized dimensions and measures (derived from 'm1' or 'd1') or the suggested ones (derived from 'm2' or 'd4') in the final query since we defined in the `WHERE` clause that suggestions are only made if no user input is available. We plan to include more complex artifacts such as subnodes of the *expression* anchestor node to support for instance computed measures.

Truncation and Ordering The node (TO) in figure 4 stands for *Truncation and Ordering*. It represents `ORDER BY` and `LIMIT` clauses of a structured query or of a certain sub-select within such a query. Thus, several nodes (TO) can occur as sub-node of a query node. If the variable '?nb' is not bound by the 'TopK' pattern, the default value as described in section V-C will be used and a single `LIMIT` will be generated. The 'Sorting Expression' (SE) representing an `ORDER BY` is not being generated in that case because the variable '?ord' is unbound. If the user entered a question starting with 'Top...' both variables '?nb' and '?ord' would be bound and we would suggest an artifact to apply the ordering (unless the user entered 'order by ...', which is parsed by a dedicated pattern). Since *top-k* questions usualy relate to a particular measure (even if the query would be 'top 5 cities'), we can safely apply the order to the recognized or suggested measure by simply relating the node for the 'Sorting Expression' (SE) to the one for the measure (MR). Note that in any case every possible interpretation with respect to the `ORDER BY` assigment would be generated.

Filters: *Filter expressions* depicted as (FE) represent a set of members or numerical values in the case of measures to be used to filter the actual result. From a data model perspective, filter expressions capture the metadata's object (either dimension or measure) on which the restriction is applied and a set or range of values that defines the actual restriction. More complex filter expressions can be defined as well (e.g. containing a sub-query). In our example, we show only examples for *member sets* (MS) containing a single member which is represented by a *value reference* (VR). In the first case, a member was directly recognized in the user's question. The variable '?dL2' originating from the dimension '?d2' is directly assigned to the *member set* and a node for the *value reference* (VR) is generated with a property for the actual value (i.e. '?vL1'). Note that we do not need to care whether the respective dimensions will be considered in the projections since this can be handled by constraints (see left part of figure 4). The second example handles personalization (e.g. "my city") and uses a filter leveraging the user profile. It works similarly as the one for matched members except that the *value reference* (VR) relates to the label of the object in the user profile that cares a similar value as one of the members of a certain dimension (e.g. 'Palo Alto' for the dimension 'City').

Range queries are conceptually similar to the ones containing a *member set*, no matter whether they are applied on dimensions or measures. The only difference is that a natural language pattern is used for detecting numeric or date range expressions in the user's question to define variables and that there are two *value references* defining the bounds of the actual *filter expression*.

As result of the mapping step, we get an RDF graph containing all potential interpretations (structured queries) of the user's question. Since the query model as such reflects the features of the underlying query language L (e.g. projections and different types of selections) it is straightforward to serialize this model to an actual *string* that can be executed on a data source. The constraints defined in previous sections ensure on the one hand how to treat different match situations and on the other hand that the generated queries are valid. The great advantage of this approach is that complex constraints can be defined in a declarative way and that they are to some extend separated from the mapping problem, making the implementation much easier in presence of complex requirements. The generated structured queries must then be scored to provide a usefull ranking of results and to define an order according to which the computed queries are eventually executed.

VII. Scoring

The previous step generates potentially several structured queries which must evenutally be ranked by relevance. To do so we combine three scoring functions by a weighted average. These are (1) a *confidence* measure that bases on text retrieval metrics, (2) the *complexity* of the generated structured query (the higher the complexity the more relevant) and (3) a measure that determines the *selectivity*, i.e. the number of queries that were generated by a certain sub-graph of the constraint definition (the less results are generated from a sub-graph the more relevant could be a query).

(1) Entity recognition confidence:. Each entity or natural language pattern match (e.g., on dimensions, measures or the top-k pattern) cares a *confidence* value c. For matched measures, dimensions or members, the *confidence* measure is a variant of the well-known *Levenshtein distance*. These values are then aggregated for a particular generated query. Let $r = \{m_1, \ldots, m_k\}$ be the query r decomposed in its k matching entities or patterns, $c_1, \ldots, c_k$ the assigned confidence values and θ_t a weight that is configured for a particular match type $t \in T$ (e.g. on dimension, measure or natural language pattern). Then the confidence $s_1(r)$ of a query r is formulated as:

$$s_1(r) = \sum_{i=0}^{k} \frac{\theta_t c_{i,t}}{k}$$

Note that the weight θ_t was experimentally determined such that $\theta_t \in [0, 1]$ and $\sum_{t \in T} \theta_t \leq 1$.

(2) Complexity: The *complexity* is a measure that bases on the number of objects used to formulate a particular query. Note that we use here all objects (including suggested ones), while we considered for *confidence* only the ones that were matches in the question. Let $r' = (count_t(r))_{t \in T}$ be the vector representing the number of entities of type t in r. Then, the complexity of a result r is defined by:

$$s_2(r) = \frac{1}{|T|} \sum_{t \in T} \theta_t r'_{i,t}$$

where $\theta_t \in (0, 1], \sum_{t \in T} \theta_t \leq 1$ is for simplicity the same weight as discussed before.

(3) Selectivity: The *selectivity* defines how specific a

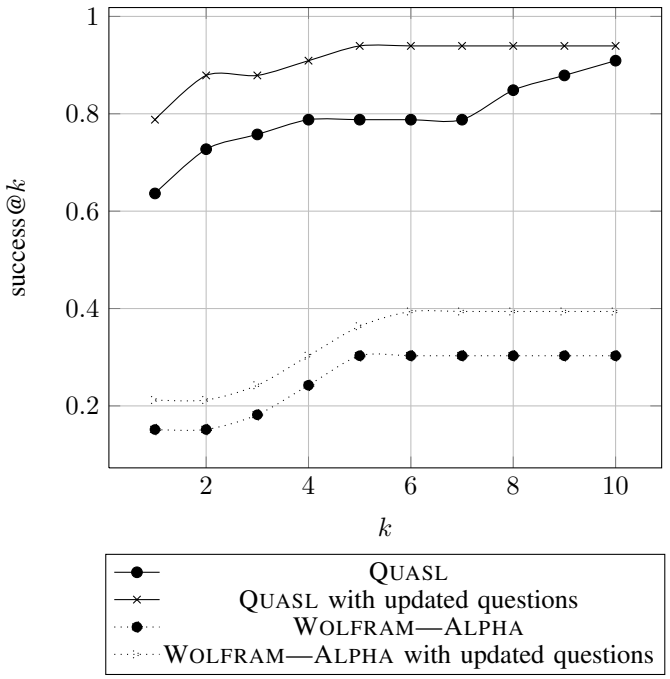


Fig. 5: Success of answering gold standard questions compared to Wolfram—Alpha

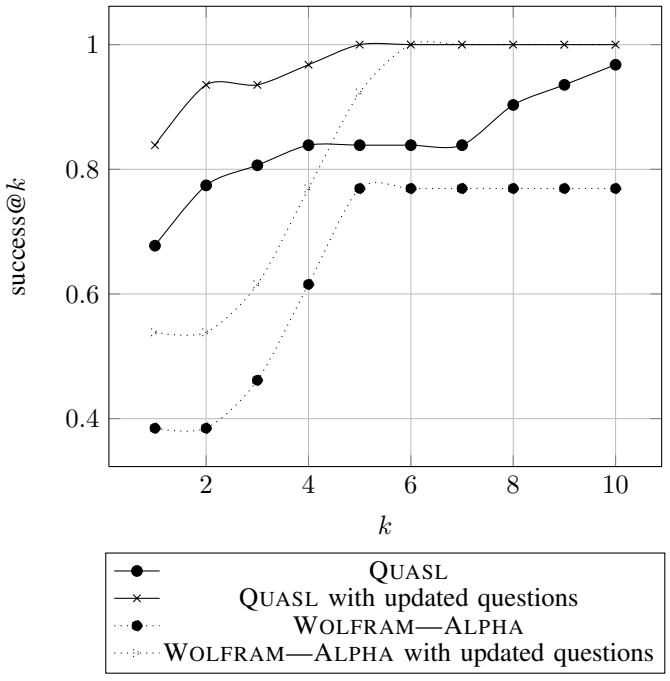


Fig. 6: Success of answering gold standard questions compared to Wolfram—Alpha when ignoring unknown data

graph-pattern is with respect to the question. The intention behind this measure is to boost very specific graph pattern matches, since they often cover special cases and are therefore usually more relevant if they occur. This "specificity" is computed as the inverse number of generated BI queries per *graph pattern* (i.e. sub-graphs of the WHERE clause). Let g be a the sub-graph in the WHERE clause that was used to compute a structured query r and $R(g,q)$ the set of all queries that where computed from the question q, given the sub-graph g. Then the *selectivity* is computed using:

$$s_3(r) = \begin{cases} \frac{1}{|R(g,q)|} & \text{if } \sigma \neq 0 \\ 0 & \text{otherwise} \end{cases}$$

where $\sigma = |R(g,q)|$ is the number of queries that have been generated by the sub-graph g.

The above mentioned scores are then aggregated by a (weighted) average, where we use $\frac{1}{3}$ for each of the parameters in our current set up. The definition of the weighting scheme is in general generic for all kinds of applications and domains. However, it remains a future task to empirical verify whether this heuristics can be applied to other domains.

VIII. Evaluation

This paper introduces a novel methodology to implement question answering systems, which is difficult to measure by quantitative metrics. However, in this section we show that one can easily implement a system that has a similar quality as a publicly available question answering system (WolframAlpha).

A. Evaluation corpus and data set

ManyEyes[8] is a collaborative platform where users publish datasets and associated visualizations (i.e. charts). One of the most popular datasets is from the American Census Bureau[9]. We have integrated it in our data warehouse and created manually the data schemas corresponding to these dataset (using the original terminology). The corpus of user questions is composed of the titles of the 50 most popular visualizations having the census dataset as data source in ManyEyes. Some of these questions are shown in table I (see last page).

B. Results

For determining the performance of our question answering system (QUASL), we use the measure *success at k* that indicates at which position k the relavant result (i.e. chart) occurs in average within the list of results. Figure 5 compares the success for $k = 1$ (on the left) to $k = 10$ (on the right) for QUASL and WolframAlpha. The results for *updated questions* stands for results where the questions has been modified in such a way that the system can better respond (see also table I). For instance, we have observed that WolframAlpha provides better results if some questions are prefixed or suffixed with "US" or "Census" to explicit a restriction to a subset of available data ("US") or to the dataset itself ("Census"). Other questions have been rephrased (e.g., "Where are the rich people" to "highest houshold income") to give the systems a chance to answer them as well. Thus the results for *updated questions* show the assumed performance when optimizing both systems with respect to the type of questions in the *gold standard* and the data set.

Given the *gold standard's* questions, our system outperforms WolframAlpha by far. However, we observed that WolframAlpha seems to include only a fraction of the Census data set. Therefore, we computed a second plot (see figure 6), which ignores all questions that could not be answered by WolframAlpha (assuming that the correpsonding data was not available to the system). Beforehand, we tried to reformulate the questions several times to ensure that the data is indeed unknown. The results in figure 6 show that our system still performs better up to $k \leq 4$ and that the performance of WolframAlpha is superior to the one of QUASL for $k > 4$ (under the assumption presented above). Thus, we can assume overall a more less similar performance of both systems.

C. Discussion

Our experiment shows that QUASL behaves to some extend similarly as WolframAlpha, which is a well-proven system. However, in the follwing we like to discuss optimizations that would further improve our performance for the given questions. The second column of table I ("updated question") depicts modification required for the system to correctly interpret users' requests. In the last column ("comment") we provide a brief explanation on how the system could be easily improved in order to correctly interpret users' initial requests.

For instance, the second question "Home ownership by State" fails, because the term 'ownership' is not part of the terminology of the data warehouse; but the term 'dewllings' appears in some measures. Thus, basic linguistic resources (like WordNet) could be used to relate synonyms or terms with similar meanings. The fifth question ("And the whitest name in America is") also requires little effort to be understood by the system: the base form of the word 'whitest' is 'white' (which is known to the system). Thus adding a stemming component would lead to an answered question in that case.

An interesting question is "Where are rich people?". It would require a little more effort in order to be correctly processed by QUASL. To answer this question, it requires to attach additional semantics to the data warhouse to determine locations that would be recognized by the term 'where'. In addition, one would configure a range filter (e.g, using natural language patterns) to declare the meaning of 'rich' (i.e. a certain income range). The question "of Americans covered by health insurance" is of similar kind, because the term "cover" can be translated to a filter on the fact table for "health insurance". The question "500MW+ Power Plants" would need a special natural language pattern to parse "500MW+".

IX. State of the art

Question Answering (Q&A) is a sub-field of Information Retrieval, and aims at delivering concise answers to queries expressed in natural language. Research in this area targeted so far mainly textual corpora. Database retrieval can be classified into two areas: Natural language interfaces (to databases) and keyword search (over databases). In the BI domain, retrieval systems go beyond executing database queries. They offer visualizations (e.g. charts) as well as reports that best suit

[8] See www-958.ibm.com/.

[9] http://www.census.gov/main/www/access.html.

Query	Updated query	Entities	Comment
State Population Change		{State, Population change}	
Home Ownership by State	Owner-occupied dwellings by state	{State, Owner-occupied dwellings}	home $\sim$ dwellings
USA States information		{State}	
Generation Y in 2010 (Ages 10-32)	population by year for ages 10-32	{Year, Population}	generation $\sim$ demographic info.
And the whitest name in America is	names white percent	{Name, White percent}	whitest $\sim$ white
40+ Population Projections by Age		{Age, Population projection}	
Average Time Spent Commuting by State		{State, Median travel time to work}	
Percent Hispanic by State		{State, Hispanic population}	
Change in city & town populations		{Town name, Population change}	
Population		{State, County, Town name, Population}	
Domestic Net Migration		{State, Domestic net migration}	
Population (by County)		{County, Population}	
US surnames	US names	{Count, Name}	surnames $\sim$ names
Age distribution by US population		{Age, Male number, Female number}	
Where are rich people?	highest household income	{State, Average household income, Median household income}	rich $\sim$ household income
People covered and not covered by Health Insurance by State		{State, Covered} Not covered}	
Southeast Asian American Population by US County		{County, Asian and Pacific islander race count}	
Dirtiest states from coal pollution	Plant name and carbon dioxide emissions	{State, Carbon dioxide emissions}	dirtiest $\sim$ emissions
50MW+ Power Plants	Plant name with nameplate capacity $>$ 500MW	{Plant name, Nameplate capacity}	500MW+ $\sim$ nameplate capacity
Emissions Per State Per Capita		{State, Methane emissions, Nitrogen oxide emissions, Mercury emissions}	
US Violent Crime		{Crime rate}	
Deaths per Year in the United States		{Cause, Per year}	
Home Valuation of Major Cities in US		{City, Min valuation, Max valuation}	
of Americans without health insurance	Americans not covered per state	{Sate, Without health insurance}	health insurance $\sim$ covered
Percent of men in Mass., 15 and over, never married		{Percent men}	
Marriages in the United States per 1,000 women by state		{State, Marriage rage per 1,000 women}	
Percentage of population aged 85+, by county		{County, Percent population aged 85+}	
Population by Age	Estimate by age	{Age, Population estimate}	population $\sim$ numeric estimate
Civilians Employed by Occupation	Amount by category	{Occupation category}	occupation $\sim$ category
Percent of Population 18+ (by state)		{State, Percent population of age 18+}	
Population categorized by ages and areas	Population categorized by ages and states	{Age, Population}	areas $\sim$ states

TABLE I: Excerpt from evaluation corpus. The symbol '$\sim$' indicates how linguistic resources could improve performances

users' queries. We discuss in the following the most important approaches in these areas and relate them to our approach.

The Q&A domain is mostly focused on "community Q&A". In this sub-domain, several users collaborate together (in a crowdsourcing way) to answer a question, and the best candidate is then chosen as the correct answer (usually by the user who asked the question in a first place). Indeed, social networks have recently played the front stage in Internet usage. In this area, several problems have arisen, like the question on how to route a question to the right expert [11], which can be seen as a classification task [12], [13]. Subjective Q&A (i.e. how to sumarize different opinions based on extracted semantics and statistics) is a topic of interest [14] as well as the prediction of questions that are not likely to be answered in the future [15]. Traditional Q&A (answering questions from a text corpus) is a huge area, but Maybury [16] outlined the various directions of Q&A systems, like their requirements (in terms of sought information), scope, complexity, etc. While traditional Q&A technologies are successful in many application areas, the algorithms used there cannot be easily applied to business use-cases with underlying databases. Indeed in this domain, data sources are mostly structured, for which keyword search approaches work best for the time beeing. Our approach tries to fill this gap by providing a methodology to develope domain and application specific question answering systems.

Keyword search over databases can be seen as a graph matching problem, where the database is represented as a graph. Recent approaches [17], [2], [3], [18] translate keyword queries into a set of structured queries to be ranked. This problem is known as the minimum Steiner tree problem, where nodes are are database entities to be associated to keywords from user's queries [19], [1], [20], [21], [22]. This kind of computation is very expensive; besides, keyword-based approaches suffer by the fact that most of the meaning of

a sentence is not conveyed by its *vocabulary* (i.e. words or *key*words in the sentence) but by the *syntax* of the sentence (i.e. its structure), as pointed out by Orsi & al. [23]. Li et al. [24] approximate this problem for answering top-k queries efficiently. Our approach tries to extend the ideas developed in this area by adding a methodology to express further semantics, e.g., for range queries or personalization.

In the BI domain, two systems have raised our attention because they are more closely related than the previous approaches. First, SODA [25] is a keyword-based search system over data warehouses. It uses some kinds of patterns to map keywords and some operators in the user's query to rules to generate SQL fragments. It integrates various knowledge sources like a domain ontology etc. However, this system does not focus on "using natural language processing to interpret the input" [25]. Our proposal is thus much more powerful, e.g., to provide means for including user context or more complicated natural language patterns and relate them with other background knowledge, which is of utermost importance as stated in [26]. Secondly, SAFE [23] is an answering system dedicated to mobile devices in the medical domain. It uses patterns, i.e. pre-defined SparQL queries with placeholders for variables. Each pattern has assigned a predefined natural language representation (i.e. a question that the user can understand) and the challenge is to rank these questions according to a keyword input posed by the user. Our approach goes beyond this idea by the ability to describe complex relations (constraints) among the recognized entities in a declarative way and map them into a structured query (which might be SQL, MDX, SparQL or any proprietary query language).

X. CONCLUSIONS

The methodology and framework presented in this paper allows developers to implement domain or application specific question answering systems in a declarative way. In addition, we elaborated in detail the application of our framework to a use case in the area of Business Intelligence, where we generate a dashboard containing charts, tables and the like to answer a user's question. The core component of our framework translates natural language input into structured queries. To this end, we have adopted a contraint-matching and mapping approach, and have implemented this approach using semantic technologies (i.e. with RDF and SPARQL). Our examples have shown that the framework is highly configurable, and therefore can be used in very different settings. The experiments showed that the use-case that was implemented with this framework has a similar performance as one of the most popular question answering systems on the Web (WolframAlpha).

The framework is not yet a product, but being evaluated with customers who provide valuable feedback. As future work we suggest two major advances. First, scoring of results can be improved by leveraging user feedback, and we like to include machine learning approaches to better rank answers (i.e. charts within a dashboard). Secondly, the system setup could be automated to some extend by providing example questions together with corresponding structured queries. To this end patterns could be automatically generated or at least suggested. Such an approach would further ease the implemention of specific question answering systems and potentially empower end-users to configure their personal system.

REFERENCES

[1] H. He, H. Wang, J. Yang, and P. S. Yu, "Blinks: ranked keyword searches on graphs," in *Proc. SIGMOD 2007*.

[2] S. Tata and G. M. Lohman, "Sqak: doing more with keywords," in *Proc. SIGMOD 2008*, pp. 889–902.

[3] T. Tran, P. Cimiano, S. Rudolph, and R. Studer, "Ontology-based interpretation of keywords for semantic search," in *Proc. ISWC/ASWC 2007*, pp. 523–536.

[4] A. Giacometti, P. Marcel, and E. Negre, "Recommending multidimensional queries," in *Proc. DAWAK 2009*.

[5] M. Golfarelli and S. Rizzi, "A methodological framework for data warehouse design," in *Proc. DOLAP 1998*, pp. 3–9.

[6] D. A. Hull, "Xerox trec-8 question answering track report," in *Proc. TREC 1999*.

[7] F. Brauer, M. Huber, G. Hackenbroich, U. Leser, F. Naumann, and W. M. Barczynski, "Graph-based concept identification and disambiguation for enterprise search," in *Proc. WWW 2010*, pp. 171–180.

[8] D. E. Appelt and B. Onyshkevych, "The common pattern specification language," in *Proc. TIPSTER 1998*, pp. 23–30.

[9] H. Cunningham, H. Cunningham, D. Maynard, D. Maynard, V. Tablan, and V. Tablan, "Jape: a java annotation patterns engine," 1999.

[10] R. Thollot, F. Brauer, W. M. Barczynski, and M.-A. Aufaure, "Text-to-query: dynamically building structured analytics to illustrate textual content," in *Proc. EDBT/ICDT 2010*.

[11] A. Pal, F. M. Harper, and J. A. Konstan, "Exploring question selection bias to identify experts and potential experts in community question answering," *ACM Trans. Inf. Syst.*, vol. 30, no. 2, pp. 10:1–10:28, 2012.

[12] T. C. Zhou, M. R. Lyu, and I. King, "A classification-based approach to question routing in community question answering," in *Proc WWW 2012*, pp. 783–790.

[13] B. Li, I. King, and M. R. Lyu, "Question routing in community question answering: putting category in its place," in *Proc. CIKM 2011*, pp. 2041–2044.

[14] T. C. Zhou, X. Si, E. Y. Chang, I. King, and M. R. Lyu, "A data-driven approach to question subjectivity identification in community question answering," in *Proc. AAAI 2012*.

[15] B. Li, T. Jin, M. R. Lyu, I. King, and B. Mak, "Analyzing and predicting question quality in community question answering services," in *Proc. WWW 2012*, pp. 775–782.

[16] M. Maybury, "New directions in question answering," in *Advances in Open Domain Question Answering*, 2006.

[17] E. Kandogan, R. Krishnamurthy, S. Raghavan, S. Vaithyanathan, and H. Zhu, "Avatar semantic search: a database approach to information retrieval," in *Proc. SIGMOD 2006*, pp. 790–792.

[18] Q. Zhou, C. Wang, M. Xiong, H. Wang, and Y. Yu, "Spark: adapting keyword query to semantic search," in *Proc. ISWC/ASWC 2007*, pp. 694–707.

[19] "Dbxplorer: A system for keyword-based search over relational databases," in *Proc. ICDE 2002*, p. 5.

[20] V. Hristidis, L. Gravano, and Y. Papakonstantinou, "Efficient ir-style keyword search over relational databases," in *Proc. VLDB 2003*, pp. 850–861.

[21] V. Hristidis and Y. Papakonstantinou, "Discover: keyword search in relational databases," in *Proc. VLDB 2002*.

[22] F. Liu, C. Yu, W. Meng, and A. Chowdhury, "Effective keyword search in relational databases," in *Proc. SIGMOD 2006*, pp. 563–574.

[23] G. Orsi, L. Tanca, and E. Zimeo, "Keyword-based, context-aware selection of natural language query patterns," in *Proc. EDBT/ICDT 2011*, pp. 189–200.

[24] G. Li, X. Zhou, J. Feng, and J. Wang, "Progressive keyword search in relational databases," in *Proc. ICDE 2009*.

[25] L. Blunschi, C. Jossen, D. Kossmann, M. Mori, and K. Stockinger, "Soda: Generating sql for business users," in *Proc. VLDB 2012*.

[26] M. A. Hearst, "'natural' search user interfaces," *Commun. ACM*, vol. 54, no. 11, pp. 60–67, Nov. 2011.