Keyword Search over Data Service Integration for Accurate Results

Vidmantas Zemleris
Vilnius University, Lithuania
(at CMS Experiment, CERN)
vidmantas.zemleris@cern.ch

Valentin Kuznetsov
Cornell University, USA
vkuznet@gmail.com

Summary

Background: Virtual data integration aims at providing a coherent interface for querying heterogeneous data sources (e.g., web services, web forms, proprietary systems) with minimum upfront integration effort.
Problem: Querying is usually carried out through a structured query language, such as SQL, which forces the users to learn the language and to get acquainted with data organization (i.e. the schema) thus negatively impacting the system's usability.
Solution: We present a keyword search system, which, given a keyword query, proposes a ranked list of structured queries along with explanations of their meanings. It operates mainly on the metadata, such as the constraints on allowed values, analysis of user queries, and only certain portions of the data. Unlike previous implementations, the system is freely available and makes no assumptions about the input query, while maintaining its ability to leverage the query's structural patterns - in case they exist.

Context: a system for Virtual Data Integration

The "CMS Data Aggregation System" (DAS):

- uses simple structured queries
- integrated access to heterogeneous services
  - → parse query & contact services
  - → eliminates inconsistencies in responses:
    - ∗ entity naming
    - ∗ data formats(XML, JSON)
  - → combines the responses
- requires only minimal service mappings
  - → no predefined schema → minimal effort in defining services
  - → complete services structure is figured out in runtime

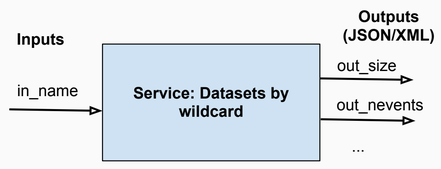


Figure 1: a data-service (simplified)

Query language must specify an entity to be retrieved and filtering criteria (as input to services). Optionally, the results could be 'piped' for further filtering, sorting or aggregation (e.g. min, avg, count, etc), e.g.:

```
dataset    dataset=*RelVal* | grep dataset.nevents >1000 | avg(dataset.size), median(dataset.size)
```

entity requested from services | conditions as service *inputs* | filters and projections on service *outputs* | aggregators

still, it is overwhelming for users to:

- learn the query language
- remember how exactly the data is structured and named

Could keyword queries solve this?

Problem definition: Interpreting Keyword Queries

Input: query, KWQ=$(kw_1, kw_2, .., kw_n)$
Task: translate it into structured query
Given: **metadata only:**

- names of entities and their attributes
  - → *service inputs* or their *output* fields
- possible values (only for some inputs)
- *constraints* on data-service *inputs*:
  - → mandatory inputs
  - → regular expressions on values

Example. Consider this query: average size of RelVal datasets with its number of events > 1000

- average RelVal dataset size nevents>1000
- avg(dataset size) RelVal "number of events">1000

For all, the expected result is:

```
dataset    dataset=*RelVal* | grep dataset.nevents >1000 | avg(dataset.size)
```

entity requested from services | conditions as service *inputs* | filters and projections on service *outputs* | aggregators

State of the art

Keyword queries:

- are ambiguous → ranked list of structured query suggestions
- nearby keywords are often related

Related Works (on keyword search over services):

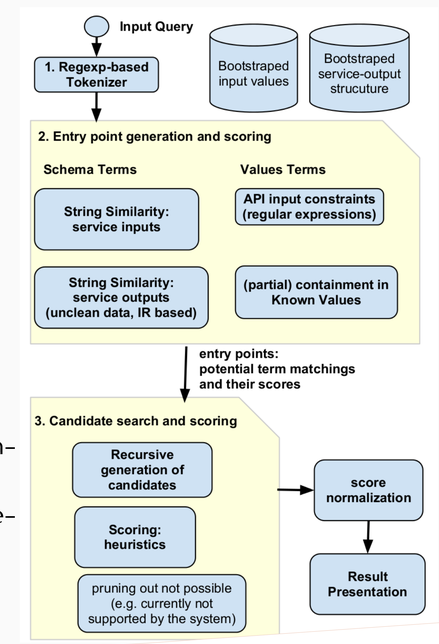
- Keymantic (the closest work)
  1. score keyword mappings individually (entry points)
     - → i.e. weights for each (keyword; domain term) combination
  2. solve *"weighted bipartite assignment"* (kw_i→domain term) problem to maximize sum of weights
     - → uses heuristics to account for keyword interdependencies
     - → solves it *approximately* with modified Munkres algorithm *with contextualizations:*
       - ∗ modifies the weights of kw_i in run time of it's "related" to earlier assignments
       - ∗ to get multiple results, repeat recursively forcing/preventing certain sub-assignments
  3. interpret generated mappings as SQL queries
- KEYRY - uses HMM (Hidden Markov Model) to label keywords
  - → HMM's initial parameters can be estimated from heuristics
  - → later machine learning can be used (if logs available)

Challenges & Solutions

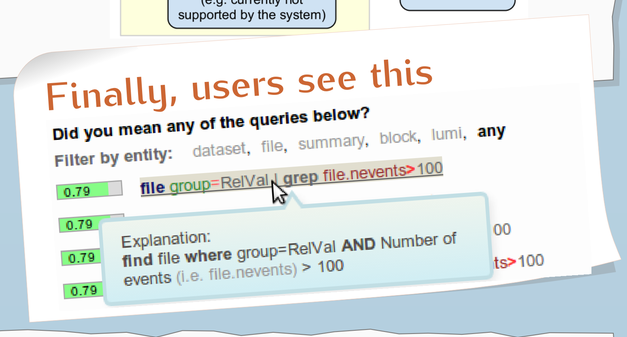
- no direct access to the data
  - → querying services is expensive → rely on metadata
  - → bootstrap list of allowed values (available only for some fields)
  - → rely on *regexps* with lower confidence (can result in false positives)
- no fully predefined schema
  - → bootstrap list of fields in service results through queries
  - → some field names are unclean (coming directly from XML, JSON responses)

Implementation overview

- *tokenizer*:
  - → clean up the query
  - → identify patterns
- identify and score "*entry points*" with
  - → string matching [for entity names]
  - → IR (IDF-based)[output fieldnames]
  - → list of known values
  - → regular expressions on allowed values
- combine *entry points*
  - → consider various *entry point* permutations (keyword labelings)
  - → promote ones respecting keyword dependencies or other heuristics
  - → interpret as structured queries



Finally, users see this



Scoring function

$$score\_prob = \sum_{i=1}^{|KWQ|} \left(\ln\left(score(tag_i|kw_i)\right) + \sum_{h_j \in H} h_j(tag_i|kw_i; tag_{i-1,..,1}) \right)$$

- $score(tag_i|kw_i)$ - likelihood of kw_i to be tag_i (from entry points step)
- $h_j(tag_i|kw_i; tag_{i-1,..,1})$ - the score boost returned by heuristic h_j given a tagging so far (often all $i-1$ tags are not needed).

Future work

- explore Ranked (Murty's) Munkres with Contextualization!?
- make more generic?

Conclusions

- keyword search over data services still lacking attention (vs. relational data-bases)
- with proper UI it may guide both beginner and advanced users
- summing log-likelihoods is better than plain scores (cf. Keymantic)

References

KEYMANTIC