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: The goal of the virtual data service integration is to provide a coherent interface for querying a number of heterogenous data sources (e.g., web services, web forms, proprietary systems, etc.) in cases where accurate results are necessary.
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. Limited access to data instances as well as users' concern with accurate results of arbitrary queries present additional challenges to traditional approaches (such as query forms, information retrieval, keyword search over relational databases) making them not applicable.
Solution: This poster presents a keyword search system which deals with the above discussed problem by operating on available information: the metadata, such as the constraints on allowed values, analysis of user queries, and certain portions of data. Given a keyword query, it proposes a ranked list of structured queries along with the explanations of their meanings. 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. The system is discussed in the context of CMS data discovery service where the simplicity and capabilities of the search interface play a crucial role in the ability of its users to satisfy their information needs.

DAS - a system for Virtual Data Integration

- uses simple structured queries
- integrated access to proprietary data-sources
  - → process the query & send requests to services
  - → eliminates inconsistencies in entity naming, data formats(XML, JSON)
  - → combining the results
- uses lightweight service mappings
  - → minimal effort in defining services
  - → complete services structure is figured out in runtime

Query language and Execution flow The queries are formed specifying the entity the user is interested in (e.g. dataset, file, etc) and providing filtering criteria (e.g. attribute=value, attribute *between* [v1, v2]). The results could be later 'piped' for further filtering, sorting or aggregation (min, max, avg, sum, count, median), e.g.:

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

- `dataset` — entity requested from services
- `dataset=*RelVal*` — conditions as service *inputs*
- `grep dataset.nevents >1000` — filters and projections on service *outputs*
- `avg(dataset.size), median(dataset.size)` — aggregators

it is overwhelming for users to

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

Can 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

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

Figure 1: a data-service (simplified)

Example. Consider these queries:

- average size of RelVal datasets with number of events more than 1000
- avg dataset size RelVal number of events>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)
```

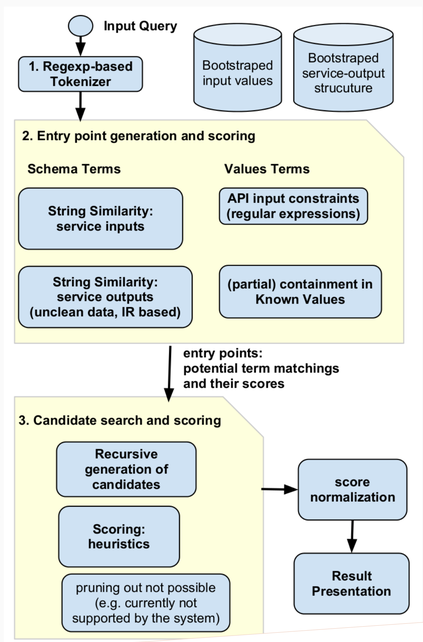
- `dataset` — entity requested from services
- `dataset=*RelVal*` — conditions as service *inputs*
- `grep dataset.nevents >1000` — filters and projections on service *outputs*
- `avg(dataset.size)` — aggregators

Existing Works

- KEYRY
- Keymantic
- SODA

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



User sees this

Did you mean any of the queries below?
Filter by entity: dataset, file, summary, block, lumi, **any**

0.79 **file** group=RelVal **grep** file.nevents>100
0.79
0.79
0.79

Explanation:
find file **where** group=RelVal **AND** Number of events (i.e. file.nevents) > 100

00
ts>100

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