# Keyword Search over Data Service Integration for Accurate Results


**Vidmantas Zemleris**
Vilnius University, Lithuania
vidmantas.zemleris@cern.ch
for the benefit of CMS Collaboration

**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 and proprietary systems) with minimum upfront effort in integration. Data is usually accessed through a structured query language, such as SQL, forcing users to learn the language and to get acquainted with data organization.
**Solution:** We present a keyword search system, which proposes a ranked list of structured queries along with explanations of their meanings. It operates mainly on the metadata, such as the constraints on inputs accepted by services. It was developed as an integral part of the CMS data discovery service focusing on simplicity and capabilities of the interface, and is currently available freely as open source.


## Context: a system for Virtual Data Integration

"CMS Data Aggregation System" (DAS):

- accepts simple structured queries
- integrates heterogeneous services
  - → parse the query & contact services
  - → eliminate inconsistencies in responses:
    - ∗ entity naming
    - ∗ data formats (XML, JSON)
  - → combine them
- requires only minimal service mappings
  - → no predefined schema
  - → minimal effort in defining services

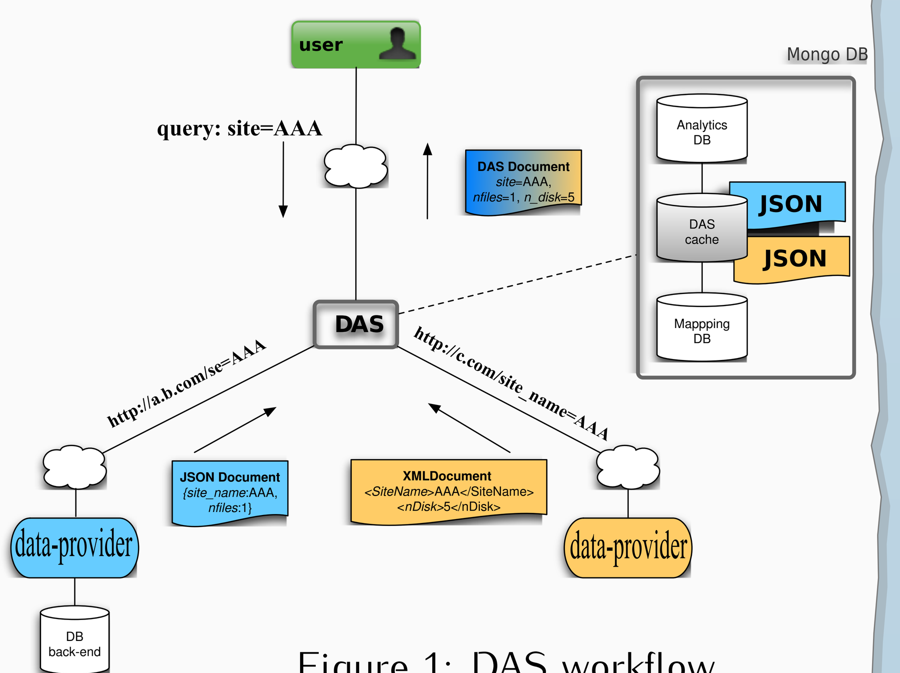
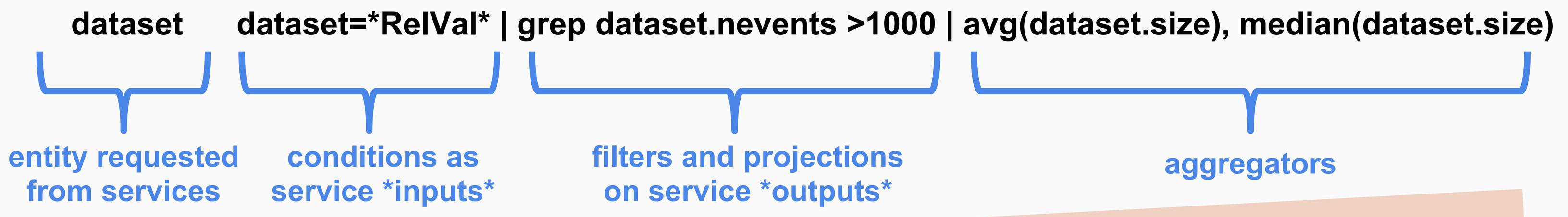


Figure 1: DAS workflow

**Queries** must specify: entity to be retrieved and filtering criteria. Optionally, the results can be further filtered, sorted or aggregated




**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?*

## Interpreting Keyword Queries: Problem definition

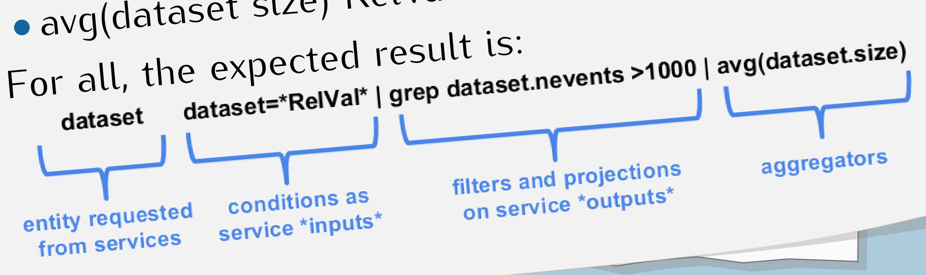
**Input:** query, KWQ=($kw_1, kw_2, .., kw_n$)
ambigous; nearby keywords are often related
**Task:** translate it into structured query
(made of $tag_j$ – schema and value terms)
**Given:** metadata only:

- 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

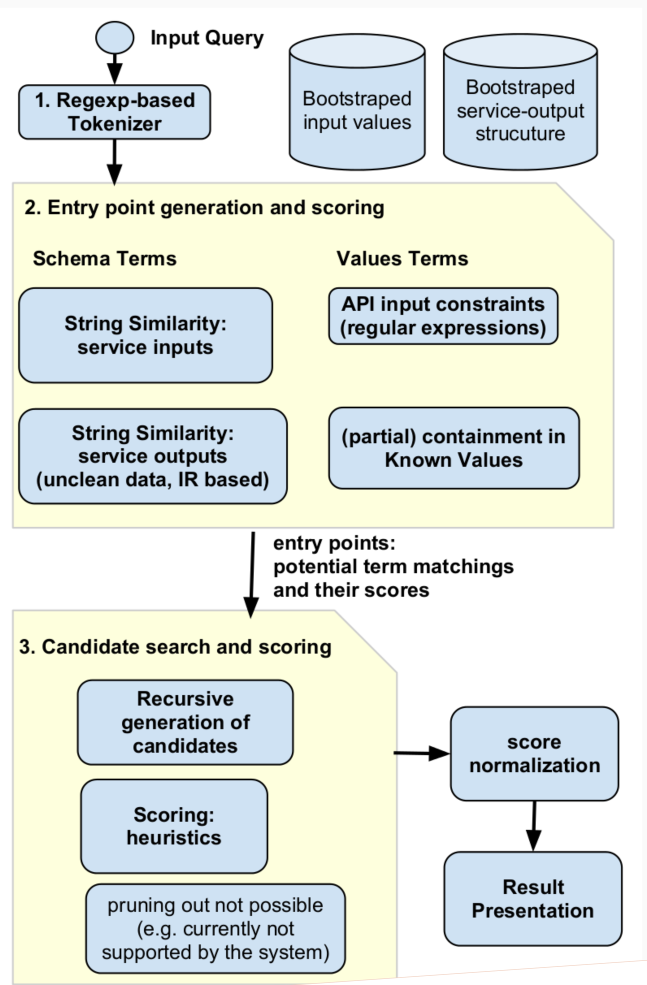
- average RelVal dataset size nevents>1000
- avg(dataset size) RelVal "number of events">1000

For all, the expected result is:

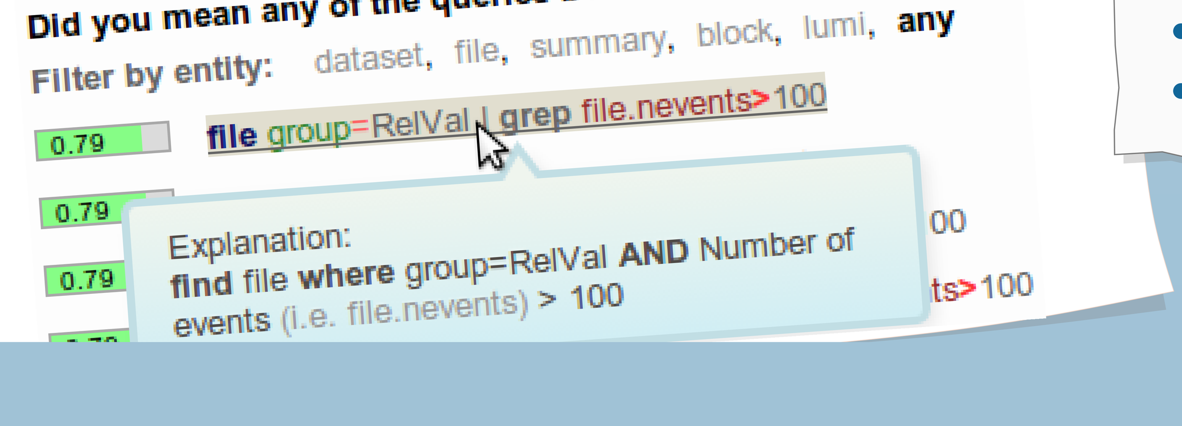



## Keyword search overview

- 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**




## Challenges & Solutions

- queries are ambiguous
  - → return ranked list of structured query suggestions
- 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 predefined schema
  - → bootstrap list of fields in service results through queries
  - → some field names are unclean → use IDF (as they come directly from JSON/XML responses)

## Details: Scoring & Ranker

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

- $score(tag_i|kw_i)$ – 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).

**So far we use exhaustive search ranker. why?**

- simpler; our schema is quite small
  - → our **cython** implementation is quite fast (often bound by MongoDB/IR engine to retrieve entry points)
- allows finding optimal solutions
- easy to filter out MANY "invalid" solution candidates that are not supported by services yet
  - → large amounts of data managed by services; performance considerations

## Related works

- Keymantic (the closest work)
  1. score keyword mappings individually (entry points)
  2. solve *"weighted bipartite assignment"* ($kw_i \rightarrow tag_j$) *with contextualizations:*
     - → maximize total sum of weights
     - → uses heuristics to account for keyword interdependencies (contextualization)
       - ∗ e.g. <table_name> <attribute>; <attribute> <its value>;
       - ∗ solves it *approximately* with Munkres algorithm modified to consider *contextualizations:*
         - · contextualize – modify weights of $kw_i \rightarrow tag_j$, if $tag_j$ is "related" to earlier sub-assignments
         - · to get multiple results, repeat recursively forcing/preventing certain sub-assignments
         - · Presenter's note: may not catch the optimum solution if:
         - · contextualization plays crucial role in selecting it
         - · and do not wish to generate ALL assignments
  3. interpret generated mappings as SQL queries
- KEYRY – uses HMM (Hidden Markov Model) to label keywords as schema terms
  - → HMM's initial parameters can be estimated from similar heuristics as above
  - → later machine learning can be used (if logs available)

## Future work & Interesting problems

- **advanced autocompletion** [photo!]
- improve the ranker
- look into advanced methods of generic performance improvements to the services like "materialized view with the incremental refresh"

**top-k (semi-)optimal solutions with contextualization?** (only ideas for now...):

- maybe Ranked (Murty's) Munkres with Contextualization!?
  - → this would at least guarantee optimal top-k for with **some** contextualization
- **out of scope, ask for handouts**

**Reflections on the HMM approach:**

- what is modelled is not same as seen by user
  - → models $kw_i \rightarrow t_j$, while user sees structured queries
  - → therefore, hard to get automatically collect training data

## 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)