Aquacold – a crowdsourced query understanding and query construction tool for the linked data web

Nick Collis and Ingo Frommholz

University of Bedfordshire, Luton, UK

Abstract. The Linked Data Web promises a disseminated, dynamic, ever expanding knowledge base where relationships between content and entities can be expressed and queried using formal logic rules. The data formats and exchange protocols that comprise the Linked Data Web have existed for over 15 years and grown to incorporate over 149 billion triples but there is no established 'mainstream' system that allows regular non-technical users to query the linked data web using natural language. In this paper we introduce *Aquacold* (Aggregated Query Understanding And Construction Over Linked Data), a novel Linked Data query tool designed to fill this gap. Aquacold combines a simple browsing interface for exploring, filtering and sorting linked data with query labelling which allows the user to store a natural language representation of the underlying SPARQL query. A search interface allows users to write guided natural language queries, which are converted into SPARQL queries to retrieve results from the linked data source. A set of voting tools are presented for each query and associated result set to surface the most accurate templates and relegate those that are less accurate. This crowdsourced approach for labelling, saving and voting on linked data query templates enables regular, non-technical users to make complex natural language queries over the linked data web without having to use a structured language such as SPARQL. In this paper we provide an overview of the system and discuss an early prototype.

1 Introduction

Linked Data has been described by Tim Berners-Lee as "The Semantic Web Done Right" [1]. The term refers to a technology stack (RDF, OWL, SPARQL) that provides a mechanism for data to be published, queried and inferred, on the web [2]. One of the promises of the semantic web is that the additional structure present in linked data should enable systems to better understand user queries and provide direct answers to questions, rather than returning links to webpages. Providing direct answers has shown to be preferable to users [3]. A critical factor preventing widespread adoption of linked data amongst regular 'non-technical' users is the lack of an intuitive user interfaces to search, sort and link this information. In other words [4]:

> ...the lack of technical knowledge and an understanding of the intricacies of the semantic technology stack limit such users in their ability to interpret and make use of the Web of Data. The key solution in overcoming this hurdle is to visualise Linked Data in a coherent and legible manner, allowing non-domain and non-technical audiences also to obtain a good understanding of its structure, and therefore implicitly compose queries, identify links between resources and intuitively discover new pieces of information

At present, in order to query the linked data web, users must either learn a formal query language such as SPARQL or rely on a narrow set of common queries which can be understood by search engines which often draw on linked data sources [5]. There is a need for a linked data search tool which can return answers from the linked data web based on natural language queries that can be written by non-technical users.

Aquacold aims to fill that gap by providing a novel combination of features that enables users to author, search and rank linked data queries represented by natural language through an efficient and simple interface. The system is designed to overcome the challenges faced by similar crowdsourced and/or programmatic approaches (see Section 5 for further details).

The paper is structured as follows. In the next section we discuss some related work. In Section 3 we introduce the Aquacold system. Section 5 reflects benefits and limitations of the current system, then we discuss future work in Section 6.

2 Related Work

The field of Linked Data search has been active in recent years, with work undertaken in a variety of different areas to increase usability and effectiveness. Natural Language Interfaces (NLI), User Interfaces (UI) and query search templates are amongst the approaches explored.

Query Builders are a subset of Linked Data search UI which enforce a strict formal syntax for users to formulate their search query. These include faceted search interfaces which use filter controls. Systems such as the CODE: Linked Data Query Wizard [6] and VisiNav [7] provide users with intuitive controls that enable queries and result sets to be manipulated in realtime. Although beneficial in terms of usability, faceted search interfaces are, by nature, limited in expressivity due to the increased level of extraction from the query itself. Each facet/filter must be predefined to control one aspect of the query so typically only small numbers of facets are made available through the UI to control the most common query elements.

Natural Language Interfaces allow users to compose their queries using regular text instead of relying on a UI to build queries or manipulate results based on predefined elements. NLIs allow for more expressive queries which are easy to understand and produce for non-expert users, however result accuracy is limited unless only basic queries are used. Few systems support more complex operations such as aggregation or comparatives.

NLI approaches for querying Linked Data range from sophisticated attempts to understand the query in its entirety as used in PowerAqua [8] and Pythia [9] to keyword based NLI used in NLP-Reduce [10] which employs a simpler 'bag of words' approach to query analysis. PowerAqua merges linked data from multiple sources and uses iterative algorithms and ranking analytics to answer natural language questions. The system performs well on large discordant data sets, but is limited in its capacity to parse more complex natural language and cannot process aggregation modifiers such as *'the most'* or *'the least'*. Pythia has stronger natural language parsing capabilities and can handle more advanced questions, including aggregation modifiers. The key limitation of the system is that it requires a manual lexicon to be constructed, which causes scalability problems.

Modern major search engines such as Google and Bing can now provide answers to natural language questions such as *'What is the capital of Egypt?'*. However, these types of simple queries form the minority of searches [11], which highlights the need for a system which can understand more complex natural language complex and return relevant answers.

Systems such as Atomate [12] and Sparklis [13] combine a UI with NLI to form a hybrid query builder / NLI that utilises the readability of the latter with the guidance of the former. Sparklis allows users to explore SPARQL endpoints through guided query construction, combining the readability of Query Builders with the granularity of faceted search, whilst retaining much of the expressivity of SPARQL. As such, queries such as *'Give me the number of pages of War and Peace'* can be built one expression at a time, requiring no knowledge of either the SPARQL language or the underlying data schema. Although a high level of accuracy is reported when testing the system against benchmark queries [14], the system does not scale well when query complexity increases. In addition, the immutable structure, wording and

syntax of the Natural Language representation of Sparklis queries does not allow for differences in expression between users and the system as a whole is not immediately intuitive and requires some time investment to learn.

Decomposing natural language queries into SPARQL using generic query templates is another approach for improving the usability of linked data search. Programmatic techniques have been developed to parse NL queries into a SPARQL template which can be used to answer user questions. TBSL [15] parses natural language to produce a SPARQL template which defines the query structure including aggregation operators such as MAX, MIN and COUNT, including variables that will be converted into URIs related to the entities in the original query. For example, the query *'Show all Universities located in the South-East of England'* would result in the following template shown in Fig. 1, which includes a variable for the class *'Universities'*, a variable for the resource *'South-East of England'* and a variable for the relation *'located in'*.

```
SELECT ?x WHERE {
?x ?p ?y .
?x rdf:type ?z
}
p: Located in
y: South-East of England
z: Universities
```

Fig. 1: Example SPARQL template used in TBSL

A number of potential queries are produced from the template, which are ranked according to string similarity, data schema conformance and prominence. Queries with a high ranking are run against the underlying triple store, with the best answer returned. TBSL answered 19 questions out of 50 correctly in the QALD-1 test. This illustrates the problem with programmatic query template generation in general - these methods produce results of insufficient accuracy to scale to 'mainstream' use, particularly with longer, more complex queries. Another limitation of such template based systems is that they are not robust enough to handle queries on rapidly evolving knowledge bases, where there are regular updates/deletions/additions.

A potential solution to this is to harness the power of the crowd. This is explored in CrowdQ [16], which uses a hybrid machine / crowd approach, producing programmatic suggestions to linked data queries, with the crowd confirming the correct answers. Although results show that the crowd could be successfully leveraged to provide structured answers to an unstructured query, the system is limited to answering short queries only and cannot process queries that involve more advanced operators such as aggregation, MIN/MAX and GROUP BY. [17] follows a similar approach, asking the crowd to pick the most appropriate URI for a given query from shortlist selected by an algorithm. One criticism leveled at this approach is that the end user is not always a subject expert and is therefore not always able to make the correct selection for the given data model and vocabulary.

Aquacold implicitly links entities in the label text with URIS and filters displayed in the data browser utilising a combination of basic pattern matching and user voting to establish the mapping. **Entity linking** (also **Named Entity Recognition and Disambiguation**) refers to entities from disparate sources that refer to the same object in the subject domain. Significant work has been done in the area of algorithmic entity analysis [18] and common challenges for such systems have been identified: [19] **Entity ambiguity** - the same string referring to multiple entities; **Name variation** - an entity which can be describe by multiple distinct words; **Absence** - text that cannot be resolved to a specific entity. By focusing on the parallel approach of crowdsourced entity linking, Aquacold avoids many of these challenges. For more details, see Section 5.

3 Aquacold – System Overview

Aquacold is a Linked Data search tool that combines a linked dataset explorer (consisting of customisable filters and a results grid), natural language input and crowdsourced SPARQL templates to enable non expert users to successfully query linked

data with no knowledge of the SPARQL query language or the underlying data schema. Aquacold employs a multi stage approach.

Fig. 2: Screenshot of the Aquacold interface with key

Figure 2 shows a screenshot of the system. Main components are a search and labelling box (section 1 and 2 in the figure) and a data browser (sections 3, 4 & 5), incorporating a results grid and set of customisable filters, which can explore a given linked data source, retrieving node labels and linking to related nodes, building up a result set based on the filters entered by the user. Once complete, the result grid and filters can be labeled using natural language (1 in the figure) with autocompletion suggestions for wording and terminology based on the labels entered by other users.

Query templates are produced based on pattern matching between queries and labels with a similar structure. These templates are used to answer natural language queries entered by users based on their similarity to labels entered by others, returning results from the linked data web to the results grid. As users compose natural language queries, they are guided using autocompletion which indicates the data available from the linked data source and the labels used to describe them. Finally, a voting system is provided to rank the results, allowing the most accurate results sets to appear first to users alongside the most accurate query labels.

It is this reciprocal combination of elements: the freedom of natural language search queries; the robustness of filtered query builders and the power of the crowd to surface relevant results; that enables non-expert users to query the linked data web. This is the novel aspect of this work. In the following we describe typical steps in the Aquacold system.

3.1 Step 1: Selecting a Linked Data source

Users begin by selecting a SPARQL endpoint for the linked data source they wish to query. At present, only one source can be explored and queried at a time, although the

system could conceivably support querying and linking from multiple data sources with further development (see Section 6). Aquacold has been successfully tested against the DBPedia, Mondial and Nobel prize endpoints.

3.2 Step 2: Building the results grid using filters

Users seed the Aquacold knowledgebase by building grids of linked data using a series of filters on subjects (e.g. *Robert DeNiro*, *Panzer Tank*, *London*) properties (e.g. *Birthdate*, *Location*, *Manufacturer*) and/or property/value pairs (e.g. *Birthdate:24/11/78*, *Location:Bedfordshire*, *Manufacturer:Vauxhall*) and labelling these grids using natural language (see Step 3). Graph and tree visualisations are often used to visualise SPARQL graphs, but these have limitations when expressing aggregation, negation or union operations, for which a grid based visualisation is preferable.

As users enter each filter (e.g. *"Starring: Pacino" or "Birthdate > 03/04/1980"*), a SPARQL query containing the filter information is run against the selected link data source (Fig. 3) the results of which populate the results grid (Fig. 4).

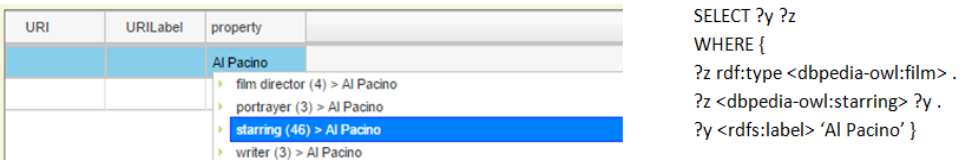

Fig. 3: Users build linked data result grids using search filters

URI	URILabel	starring
http://dbpedi...	...And Justice for All (film)	John Forsythe, John Forsythe, Lee Strasberg, Lee Strasberg, Al Pacino, ...
http://dbpedi...	Author! Author! (film)	Tuesday Weld, Dyan Cannon, Bob Dishy, Al Pacino, Alan King (comedian...
http://dbpedi...	Glengarry Glen Ross (film)	Jack Lemmon, Kevin Spacey, Alan Arkin, Jonathan Pryce, Al Pacino, Alec...

Fig. 4: The query is run against the linked data cloud and results are returned

Additional filters can then be added, or **multiple criteria** can be added to existing filters. More advanced operators such as AND / OR / MAX / MIN (Fig 5) are under development. Each filter highlights valid values that can be entered as the user

URI	URILabel	starring	producer
		Al Pacino AND Robert DeNiro	
http://dbpedi...	The Godfather Saga	Marlon Brando, Robert Duvall, Robert De Niro, John...	Albert S. Ru...
http://dbpedi...	Heat (1995 film)	Jon Voight, Tom Sizemore, Ted Levine, Robert De N...	Matthias Go...
http://dbpedi...	The Godfather (film series)	Eli Wallach, Marlon Brando, Robert Duvall, Sofia Co...	Fred Fuchs,...

Fig. 5: Operators such as AND, OR, MIN, MAX, AVERAGE can be used to expand the query

types using autocompletion suggestions (Fig. 6). Invalid values are not permitted. Basic fuzzy text matching, using the regex feature in SPARQL 1.1, is used to match text entered by the user with available values from the linked data set. This approach has its limitations, and could be improved with the addition of more sophisticated entity recognition techniques in a future version (see the discussion in Section 6). Different filter operators are provided depending on the filter datatype. This is retrieved

using the SPARQL 1.1 datatype function. For example, if the datatype is *xsd:integer* operators such as $<$ and $>$ can be used and if the datatype is *xsd:dateTime* only valid dateTime values will be permitted.

In the examples above, each result row is formed of one linked data node that has properties (e.g. *label, starring, producer*) linked directly to it. This is sufficient for simple queries, but the real value of linked data is in linking subgraphs and the more complex queries that can be performed based on this linking. Aquacold filters allow users to link nodes that share related properties, eg *"starring actors born in Italy"*. To create such a filter, the user would enter *"Starring: * Italy"* in the filter definition, the asterisk wildcard character instructs Aquacold to search for all labels matching the text Italy directly linked to nodes that are connected to the original node by the *starring* property.

Employing a linked data browsing UI such as this helps non-expert users explore linked data without having to code SPARQL and ensures the underlying SPARQL code formed by the UI is produced in a consistent format, which is necessary for the template generation (see Subsection 3.4) to work correctly.

3.3 Step 3: Labeling the results grid

Whilst the filters are being defined, Aquacold offers suggestions for labels to describe the result set. These suggestions are based on the labels that other users have entered who have used the same filters as the current user. For example, if other users who used the filters *'starring: Al Pacino'* and *'starring: Robert DeNiro'* had labelled their results *'Films starring Al Pacino and Robert DeNiro'*, this would be offered by the system as a label suggestion when a user adds the filters *'starring: Al Pacino'* and *'starring: Robert DeNiro'* to their results grid (Fig. 7). The order in which the suggestions are presented is dictated by the score assigned by user voting (see *step 6*).

3.4 Step 4: Template generation

This is a back-end process that is not visible to the user. When a new label is created, the label and associated SPARQL code generated by the results grid are saved to the Aquacold database. Aquacold then compares the label – and the associated SPARQL query – with labels and templates on the database. Where partial matches are found that contain unmatched elements at the same location in both the labels and the SPARQL query, a template query is created and stored (Fig. 8). Each template query contains the same SPARQL structure as the queries on which it is based, but replaces the unmatched entity with a variable data binding. Each corresponding label contains the same text as the original title but replaces the unmatched entity with a wildcard e.g. [ACTOR].

3.5 Step 5: Searching linked data

The search component of the Aquacold system is powered by the query templates (which are in turn powered by the data browsing and result grid labelling components). Users enter their search query in natural language, then Aquacold will

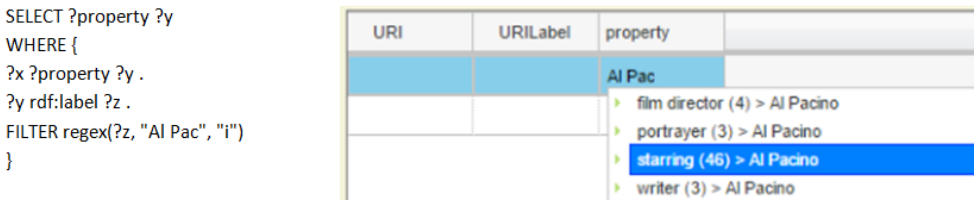

Fig. 6: Example of partial match autocompletion in a filter

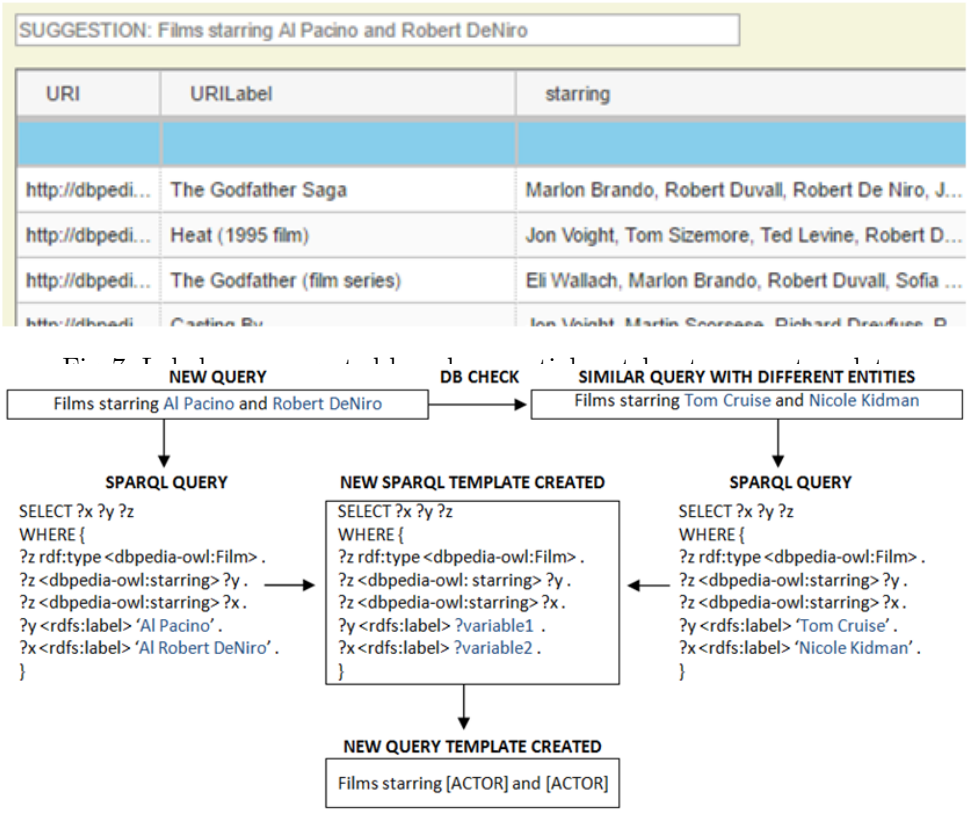

Fig. 8: Query template generation process

attempt to match the query against the template database, replacing the variables present in the template with entities from the query that appear in place of the wildcards. Aquacold will then run this query against the linked data source to retrieve the results. Whilst constructing their query, users are guided with suggestions of

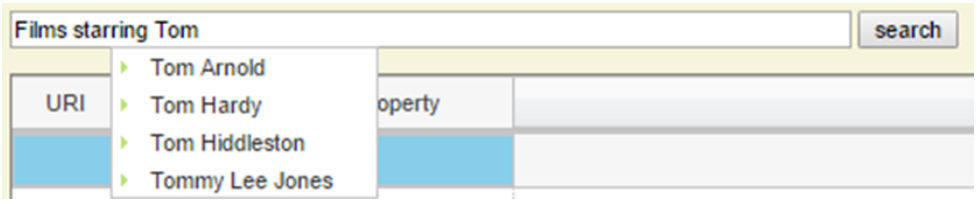

Fig. 9: Entity autocompletion suggestions

potential words and phrases based on labels entered by other users. When a user's query matches a label template up until the wildcard portion of the label, Aquacold will retrieve all possible URIs where the *rdf:label* value matches the text the user has entered for the wildcard.

In the example above, the user begins by entering the text *'Films starring'*, which matches the template **'Films starring [ACTOR]'** up until the wildcard **[ACTOR]**. This triggers the query autocompletion component, retrieving all entities from the linked data source that match a) the constraints applied by the SPARQL template (in this instance, they must exist as a subject of the property *<dbpedia-owl:starring>*) and b) the text the user has entered so far – in this instance an *rdf:label* that contains the string *'Tom'* (see Figure 10).

If the user continues to match the template **'Films starring [ACTOR] and [ACTOR]'** up until the second [ACTOR] they will again be prompted with an autocomplete option for the second actor and so on.

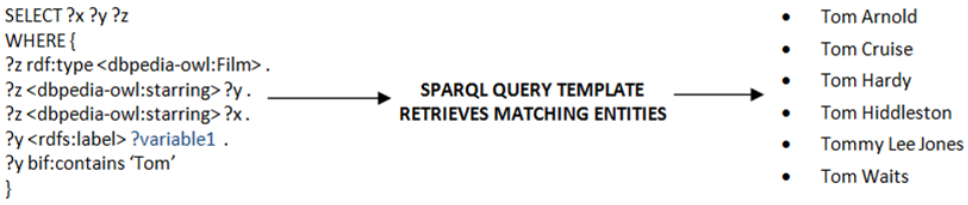

Fig. 10: Entity autocompletion SPARQL

When users submit a query, the grid interface is populated with the results. A message is displayed to the user if no results are returned. Provided results are returned, the grid can then be further manipulated to expand on the existing results sets (e.g. adding a filter *director:MartinScorsese*) and an updated label can assigned accordingly (e.g. **Films starring Al Pacino and Robert DeNiro directed by Martin Scorcese**).

3.6 Step 6: Voting

A vote up / vote down interface is provided to users to rank result sets returned by queries, improving the visibility of accurate templates and relegating those that are less successful. Consider the following scenario: *User A* could create a result grid returning all Grand National Winners since 1901 and apply the label **'Grand National Winners since 1901'**. However, they make a mistake and apply the *date > 1910* filter instead of *date > 1901*. *User B* then searches for **'Grand National Winners since 1901'** and receives the incorrect results that *user A* defined. *User B* corrects the filter and saves the grid with the same label **'Grand National Winners since 1901'**. Next, *User C* arrives at the site and makes the same query. They are presented with both result sets created by *user A* and *user B*. On viewing both, *user C* can see the mistake made by *user A* and votes down their version of the results set (giving it a score of -1) whilst voting up the correct version made by *user B* (giving it a score of +1). Now, when subsequent users make the same query, they are shown *user B*'s version initially as it has a higher score. This is a simple example that highlights how user voting can be used to surface preferred result sets when many are available. Future updates could introduce more sophisticated voting and ranking mechanisms.

4 Details of the AQUACOLD Interaction process

Consider the following scenario as an example of this ecosystem (*Fig 11*):

1. **User A** arrives at the site looking for information on the type of tanks that were used in World War 1. He/she enters *'Tanks used in World War 1'* into the AQUACOLD search box. No results are found for this query.
2. Unable to find any pre-existing results, **User A** builds the results grid themselves by adding the filters *dbpedia:type = :Tank* and *dbpedia:usedInWar = :worldWar1*. As each filter is applied, the grid is populated with results.
3. The results grid complete, **User A** adds the label *'Tanks used in World War 1'*.
4. Aquacold populates the database with **User A's** query label and the associated SPARQL code from the results grid, together with possible variations formed by replacing all entities in the query with wildcards *e.g. Tanks used in [*]*.
5. Another user, **User B** searches for *'Tanks used in World War 2'*. Although a result grid for this query has not been explicitly created by another user, results are returned by the template created in step 4, substituting the entity *'World War 1'* with *'World War 2'*.

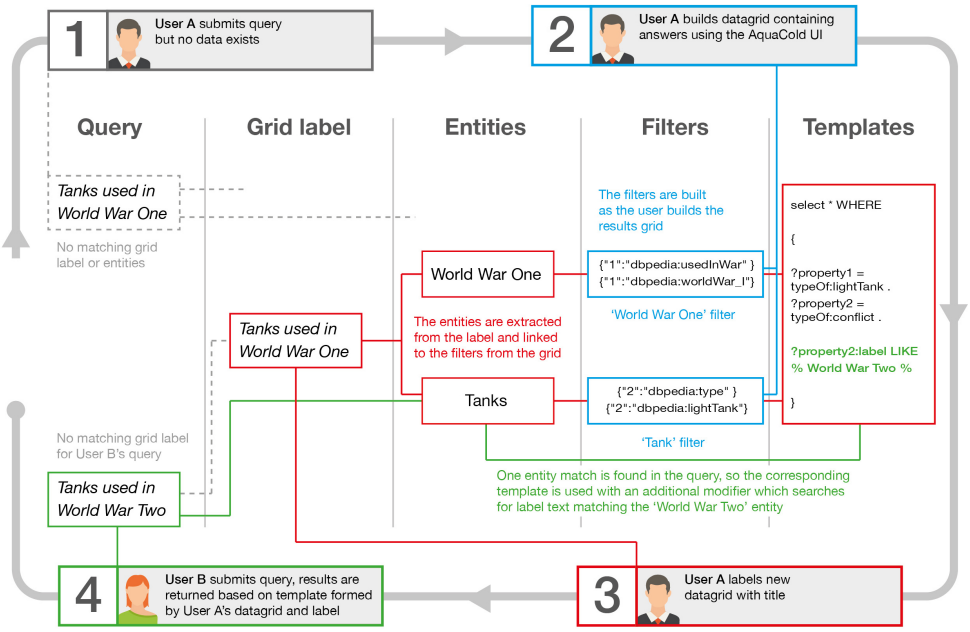

Fig. 11: Users interacting with the Aquacold ecosystem

5 Benefits and Limitations

5.1 Benefits

The Aquacold ecosystem of result grid construction through filtering, result grid labelling, template generation, guided query construction and result ranking through voting can continue indefinitely, providing an effective and easy to use search tool for a wide range of queries over a linked data source which requires no knowledge of SPARQL or the underlying data schema. As more people use the system and the volume of query templates and user votes in the database increases, higher quality results should be available for queries of increasing complexity, which in turn should drive more people to use the system, which will in turn increase volume and quality, and so forth.

Aquacold offers a number of benefits over comparable linked data search systems. Free natural language can be used to label Aquacold queries, enabling greater expressivity than systems such as SPARKLIS [13] which rely on guided query construction using rigid, formal vocabularies, which can result in overly verbose labels which may not reflect a user's preferred description. E.g. the user may wish to express their query as *'How many languages are spoken in Columbia?'* whereas SPARKLIS enforces *'give me every language spoken in Colombia and give me the number of language'*. With Aquacold users can define their own labels, with crowdsourced voting tools promoting quality control.

By employing a crowdsourced approach for translating between natural language and SPARQL, more flexibility is provided than using algorithmic methods such Quepy [20], allowing synonyms, slang and multiple language variations to be recognised. Once a user has labeled a results grid with a particular term, it is instantly available for other users to access.

Aquacold incorporates a similar filter controlled results grid to the CODE: Linked Data Query Wizard [6], but offers several benefits by combining this with a natural language interface for labeling result sets. A primary benefit is improved discoverability of results by allowing users to relabel existing results sets and vote on which of the alternative labels is more accurate. With systems such as CODE, query labels are set by the original result set author and are immutable.

They also require results sets to be labeled individually, e.g. *"All books written by Stephen King"*, which limits the utility as a search tool by requiring thousands

of labels to be manually assigned before a dataset of sufficient size is available. A key benefit of the Aquacold template system is that labels and associated results sets will be automatically generated for all related queries e.g. *"All books written by Peter Straub", "All books by Michael Crichton" etc*, significantly increasing the rate at which the database is populated by queries and in turn, its utility as a search tool.

The crowdsourced approach employed by Aquacold avoids the inherent challenges of algorithmic entity recognition (as identified by [19]) **Entity ambiguity** - the same string referring to multiple entities; **Name variation** - an entity which can be describe by multiple distinct words; **Absence** - text that cannot be resolved to a specific entity;

Entity ambiguity is handled through free text labeling and user voting. Consider the label *"Timezone in Birmingham"*. The entity Birmingham is ambiguous as it may refer to both Birmingham, UK or Birmingham, Alabama (and potentially other locations). By allowing separate result grids to be created by users that contain the timezone details of both locations, and allowing the same label *"Timezone in Birmingham"* to be used for both, user voting can ensure that the most popular location appears at the top of the list, with alternatives easily selectable. The crowdsourced approach enables **name variation** to be handled similarly. Consider the label *"Films starring Robert DeNiro"*. Aquacold allows users to apply multiple labels to the same result grid, eg *"Films starring DeNiro"*, *"Films starring Bob DeNiro"*. Voting would again be used to surface the most popular variation. In cases of **absence** - where labels cannot be resolved to a specific entity - users are able to use the data browsing tools to build their own results set and apply their own labels (see *Fig 11*). The chances of this happening are mitigated through the use of guided query construction, which suggests valid, recognised terms to the user as they construct their query.

Current crowdsourcing approaches to formulating and answering queries over linked data [21,16] have several limitations. Many used paid microservice platforms such as Amazon's *Mechanical Turk* to source, manage and pay 'workers' for their contributions. This often results in biased results [21] that indicate many workers favour options that are quick to select and do not bother to read all options available. There is also the cost of using microservices for crowdsourcing to consider, which can be significant when large datasets are used.

AquaCold avoids these issues of bias and cost present in paid-for crowdsourcing as users have an intrinsic incentive to use the system, rather than a profit motive. After querying the system, if users receive results they believe to be incorrect, they can adjust the results grid using the data browsing interface to correct the results set and resave it. This will store the updated results set as an alternative answer to the original query which will be accessible to all users and can be voted up or down accordingly. Aquacold offers a sustainable ecosystem of data discovery, query construction, query refinement, labeling, searching and voting that results in a organic and scalable incentive for users to engage with the system.

A further advantage of Aquacold's unrestrained labeling system is that queries of any size and complexity can be answered, whereas systems that use a programmatic approach are more limited. Even CrowdQ [16] which uses the crowd for query understanding is limited to answering short queries only.

It is hypothesised that Aquacold's human labelled result sets and crowdsourced quality control will result in higher precision compared with systems which use programmatic translation of Natural Language Queries to retrieve linked data results.

5.2 Limitations

Aquacold is early in development and faces a number of challenges to expand beyond the testing phase.

Due to the reliance on user generated queries for template seeding, Aquacold is unable to produce results from a 'cold start', therefore the usefulness of the system is limited initially until sufficient data has been seeded by users. This is a limitation when compared to pure NLI approaches such as PowerAqua [8]. This *'cold start'* problem is faced by all crowdsourced systems. The amount of time required for manual curation and annotation is significant. One potential solution may be to analyse logs of existing search queries (as explored in [16]) and ingest these into the system by means of an automated or semi-automated process.

Trust is a related challenge shared by all systems that rely on crowdsourcing. The simple vote up +1, vote down -1 system employed by Aquacold is sufficient for an early proof of concept stage, but more sophisticated methods will be required to account for issues of trust, prevent manipulation and incorporate different levels of user rights. These issues are common for all systems that involve open curation of a knowledgebase such as Wikipedia.

It could be argued that a limitation of allowing users free reign in labelling result sets in Aquacold is that multiple different labels can be attached to the same query which increasing ambiguity, whereas the formalised structure employed by guided query construction tools such as SPARKLIS [13] ensures a 1:1 relationship between labels and queries. The use of crowdsourced tools goes some way towards ameliorating this however, as the most accurate labels will be surfaced by the crowd over time.

At present Aquacold works with a single linked data source only. Most of the testing to this stage has been completed using DBpedia, however the system will work with any SPARQL endpoint, provided the entities have an associated *rdf:label* property. This reliance on one property type is a further limitation of the system. Aquacold could be expanded to incorporate other label types to address this deficiency.

Although the voting system should provide some level of quality control, there is a danger that the same issues with disparity that affect existing linked data ontologies could be transferred to the Aquacold system. Consider the query *'Nobel prize winners from 1989 to 1998'*. This label could be applied to a result grid containing this data from the DBPedia linked data source. However, the same label containing the same results could also be built from the *data.nobelprize.org* data source. As both contain identical information, user voting would not result in the more accurate result set being surfaced. One potential remedy that could be applied if Aquacold was developed to work with multiple linked data sources (see *Future work*), would be to include an equivalency function such as *owl:sameAs* in the filters to define a direct relation between the URIs from different linked data sources. To support cross domain queries that link between different data sources, some redevelopment is required (see *Future Work*).

The filters used to build result grids have a number of limitations. Fuzzy text matching is used to match the text a user enters in the filter box with a linked data node's *rdf:label* property. This is inherently limited as labels that do not contain the exact string that users enter are not returned. There is no allowance for polysemy (multiple meanings of a word), potential typos the user may make or related words that may be relevant.

Although text based filtering is supported, more advanced filtering on datatype specific operators such as <> for integers and BETWEEN expressions for dates, or OPTIONAL elements, UNIONS and CONTRUCTS are not yet possible, which is a limitation when compared with tools such as TBSL [15]. There are plans to support these features in a later version of Aquacold. No matter how feature rich the filter UI becomes, as a Query Builder interface, it will never match the expressivity possible in a SPARQL query.

6 Conclusion and Future Work

Aquacold exists as an early prototype and has yet to be tested. The next stage of development will be to compare effectiveness against similar systems, Aquacold will be evaluated using questions from the QALD [22] challenge, an established benchmark for comparing and assessing NLI interfaces for linked data. Many comparable systems such as SPARKLIS [13], Crowdq [23] and TBSL [24] have employed questions from the QALD challenge to measure their effectiveness.

This measure will be broken down into several metrics including: **Expressivity** - The proportion of questions that can be answered successfully; **Scalability** - The total interaction time to retrieve successful answers and **Usability** - How the participants found using the system (e.g. how easy or difficult they found getting answers). Each metric will evaluated by participants with a range of technical abilities and subject knowledge.

At present, Aquacold works with one linked data source at a time, but could be developed to enable query answering and query construction across multiple linked data sources that can be connected in the same way as connections to related linked data nodes are made within the current system. Once this development is completed, data from an additional datasets can be incorporated into the results grid where a object URI is shared between the two. Further improvements could also be made to the filters to enable them to produce queries using more advanced elements of the SPARQL language such as OPTIONAL elements, UNIONS and CONTRUCTS.

Another avenue to explore for future development is to augment Aquacold with external tools such as Silk [25] which could be used to discover related entities from different data sources. This could be combined with interlinking to enable filters to interlink related entities without them being explicitly defined.

References

1. Berners-Lee, T.: The Great Unveiling (2009)
2. W3: W3c Ontology definition. (2015)
3. Bernstein, M.S., Teevan, J., Dumais, S., Liebling, D., Horvitz, E.: Direct answers for search queries in the long tail. Proceedings of the 2012 ACM annual conference on Human Factors in Computing Systems (2012) 237–246
4. Dadzie, A.s., Rowe, M.: Approaches to Visualising Linked Data: A Survey | www.semantic-web-journal.net. **1** (2011) 1–2
5. Chilton, L.B., Teevan, J.: Addressing People ' s Information Needs Directly in a Web Search Result Page. Www2011 (2011) 27–36
6. Hoefler, P., Granitzer, M., Veas, E., Seifert, C.: Linked data query wizard: A novel interface for accessing sparql endpoints. CEUR Workshop Proceedings **1184** (2014)
7. Harth, A.: VisiNav: A system for visual search and navigation on web data. Journal of Web Semantics **8**(4) (2010) 348–354
8. Lopez, V., Fernández, M., Stieler, N., Motta, E., Hall, W., Mkaa, M.K., Kingdom, U.: PowerAqua : supporting users in querying and exploring the Semantic Web content. (2011)
9. Unger, C., Cimiano, P.: Pythia: Compositional meaning construction for ontology-based question answering on the semantic web. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **6716 LNCS** (2011) 153–160
10. Kaufmann, E., Bernstein, A.: Evaluating the usability of natural language query languages and interfaces to Semantic Web knowledge bases. J. Web Sem. **8**(4) (2010) 377–393
11. White, R.W., Bilenko, M., Cucerzan, S.: Studying the Use of Popular Destinations to Enhance Web Search Interaction. Proceedings of the 30th annual international ACM SIGIR conference on Research and development in information retrieval - SIGIR '07 (2007) 159–166
12. Van, M., Kleek, M.V., Moore, B., Karger, D.: heterogeneous information sources on the web Citation Accessed Citable Link Detailed Terms Atomate It ! End-user Context-Sensitive Automation using Heterogeneous Information Sources on the Web. (2017)

13. Track, D.: Sparklis : a SPARQL Endpoint Explorer for Expressive Question Answering. (2015)
14. Cabrio, E., Cimiano, P., Lopez, V., Ngomo, A.C.N., Unger, C., Walter, S.: QALD-4: Multilingual question answering over linked data. CEUR Workshop Proceedings **1179** (2013) 1172–1180
15. Walter, S., Unger, C., Cimiano, P., Daniel, B.: Evaluation of a layered approach to question answering over linked data. (00) (2012)
16. Demartini, G., Kraska, B., Franklin, M.: CrowdQ: Crowdsourced Query Understanding. Conference on Innovative Data Systems Research (CIDR) (2013) 4
17. Damljanovic, D., Agatonovic, M., Cunningham, H.: FREyA: An interactive way of querying linked data using natural language. Lecture Notes in Computer Science (including subseries Lecture Notes in Artificial Intelligence and Lecture Notes in Bioinformatics) **7117 LNCS** (2012) 125–138
18. Shen, W., Wang, J., Han, J.: Entity linking with a knowledge base: Issues, techniques, and solutions. IEEE Transactions on Knowledge and Data Engineering **27**(2) (2015) 443–460
19. Wu, H.j.D.C.y., Tsai, R.T.h.: From Entity Recognition to Entity Linking : A Survey of Advanced Entity Linking Techniques. The 26th Annual Conference of the Japanese Society for Artitifical Intelligence (2012) 1–10
20. Bansal, R., Chawla, S.: An Approach for Semantic Information Retrieval from Ontology in Computer Science Domain Query for. (2) (2014) 58–65
21. Damljanovic, D., Petrak, J., Lupu, M., Cunningham, H., Carlsson, M., Engstrom, G., Andersson, B.: Random Indexing for Finding Similar Nodes within Large RDF graphs. (2012) 1–15
22. Unger, C., Forascu, C., Lopez, V., Ngonga Ngomo, A.C., Cabrio, E., Cimiano, P., Walter, S.: Question Answering over Linked Data (QALD-4). Nicola Ferro (2014)
23. Haas, D., Bruckner, D., Harper, J.: CrowdQ : A Search Engine with Crowdsourced Query Understanding
24. Unger, C., Lehmann, J., Gerber, D.: Template-based Question Answering over RDF Data. (2012) 639–648
25. Volz, J., Bizer, C., Gaedke, M., Kobilarov, G.: Silk – A Link Discovery Framework for the Web of Data. (2009)