A SPARQL engine for crowdsourcing query processing using microtasks

Maribel Acosta
Institute AIFB
Karlsruhe Institute of Technology, Germany
maribel.acosta@kit.edu

Elena Simperl
Web Science and Internet Research Group
University of Southampton, United Kingdom
E.Simperl@soton.ac.uk

Fabian Flöck
Institute AIFB
Karlsruhe Institute of Technology, Germany
fabian.floeck@kit.edu

Barry Norton
Ontotext AD, Bulgaria
barry.norton@ontotext.org

ABSTRACT

There are queries in Linked Data processing that cannot always be optimally answered through traditional data base management techniques. More often than not answering such queries relies on information that is incomplete, incorrect, or fuzzily specified; and on mere approximations of computationally advanced functionality for matching, aggregating, and ranking such information. As a means to deal with these limitations, we propose CrowdSPARQL, a novel approach to SPARQL query answering that brings together machine- and human-driven capabilities. We define extensions of the SPARQL query language and the Linked Data vocabulary VoID in order to capture those aspects of Linked Data query processing that per design are likely to benefit from the use of human-based computation. Based on this information, and on a set of statistics gathered during the use of our system, CrowdSPARQL is able to decide at run time which parts of a query are going to be evaluated using automatic query execution techniques, and which will be answered by the crowd via a microtask platform such as Amazon's Mechanical Turk. We evaluated CrowdSPARQL in a scenario handling a representative subset of tasks that are amenable to crowdsourcing - ontological classification, entity resolution and subjective rankings - on the DBpedia and MusicBrainz data sets, in order to learn how specific parameters of microtask design influence the success of crowdsourced query answering.

1. INTRODUCTION

The term 'microtask crowdsourcing' refers to a problem-solving approach, by which a problem is decomposed into simple tasks that can be solved largely independently by a distributed group of people. Through platforms such as Amazon's Mechanical Turk[1] and CrowdFlower[2] the idea has proven successful in a multitude of scenarios, in particular when it comes to the resolution of those kinds of task which can hardly be tackled in a fully automated fashion. There are numerous examples of such tasks, from finding and reconciliating information on the Web or in corporate data bases; to producing, revising, and translating text; transcribing audio content; and labeling images. Increasingly, crowdsourcing platforms are also being exploited by researchers, in particular in extension to automatic methods and techniques addressing the types of tasks just mentioned, in fields as diverse as Information Retrieval, Information Extraction, Multimedia Processing, and Data Base Management Systems.

In previous work of ours we proposed a general architecture in which microtask crowdsourcing becomes an integral part of Linked Data technology [**?**, **?**]. Within this architecture the core building blocks that are required to build and run Linked Data applications, and their decompositions as microtasks, are described declaratively. This facilitates the automatic design and seamless provisioning of crowdsourcing functionality that complements those parts of a Linked Data application that operate primarily in an automatic fashion. The functionality of established Linked Data technology components - for instance, RDFication, data curation, data interlinking and query answering - is expanded with means to interact with microtask platforms such as Mechanical Turk (MTurk) to post specific HITs (the unit of work on MTurk); assess the quality of the inputs obtained from the crowd; and exploit the resulting RDF in a particular application. The interaction with the microtask platform can occur offline, when crowdsourcing input is being used to gradually improve the quality of computational techniques, and online, which may require additional optimizations to predict the time-to-completion of crowdsourced tasks [**?**] and the accuracy of their outcomes [**?**].

In this paper we present CrowdSPARQL, an implementation of this hybrid Linked Data architecture for SPARQL query answering. We define extensions of the SPARQL query language and the Linked Data vocabulary VoID in order to capture those aspects of Linked Data query processing that per design are likely to benefit from the use of human-based computation. Based on this information, and on a set of statistics gathered during the use of the query engine, CrowdSPARQL is able to decide at run time which parts of

[1] `http://www.mturk.com`

[2] `http://www.crowdflower.com`

Permission to make digital or hard copies of all or part of this work for personal or classroom use is granted without fee provided that copies are not made or distributed for profit or commercial advantage and that copies bear this notice and the full citation on the first page. To copy otherwise, to republish, to post on servers or to redistribute to lists, requires prior specific permission and/or a fee.
.

a query are going to be evaluated using automatic query execution techniques, and which will be answered by the crowd via Mechanical Turk.

2. MOTIVATION

There are several reasons why we believe Linked Data query answering could and should be extended into microtask crowdsourcing. A great majority of queries in Linked Data processing cannot always be optimally answered through traditional data base management techniques; this situation is the result of a number of factors, which we will briefly explain in the following.

Two of the primary advantages claimed for exposing data according to the Linked Data principles are improvements and uniformity in data discovery and integration at Web scale. In the former case a 'follow-your-nose' approach is enabled, wherein links between data sets facilitate browsing through the Web of Data. At the technical level previously undiscovered data is aggregated, and enriches the semantics of known resources (ad-hoc integration), by virtue of the RDF's uniform data model. True integration across this Web of Data, however, is hampered by the 'publish first, refine later' philosophy encouraged by the Linking Open Data movement. While this philosophy is surely one of the main reasons behind the impressive growth we have seen in the amount of Linked Data online over the past five years, the quality of this data and of the links connecting data sets, and the sparsity of the linkage,[3] is something that the community is often left to resolve. In particular the gap between informal browsing and effective queries, which require properly aggregated data, has been pointed out in recent work.[5] Despite promising technology groundwork through standardized representation formats and access protocols, the actual experience with developing applications that consume Linked Data soon reveals the fact that, for many components of a Linked Data applications this is hardly the case. While this might be understood as a temporary situation, which could be solved once the technology becomes more mature, the history of data integration shows that the core task indicates the opposite. The question of whether two entities are the same or related is often heavily relying on contextual or domain knowledge, and thus on human input.

Another aspect of the Linked Data publication process which leads to undesired effects when applications attempt to consume the resulting data is the tendency to shift the emphasis from the classification of resources according to specifically targeted categories and vocabularies, to the use of generic, broadly purposed vocabularies, and interlinking of resources using standard properties such as identity and general relatedness. This implies that more often than not, when answering a SPARQL query, it is not possible to infer classification from the properties in used by applying established reasoning techniques.

A third source of inaccuracy in answering queries over Linked Data are optional constructs in SPARQL, whose optimal evaluation requires domain- or application-specific information, which is not usually contained in the published data sets. The most obvious example in this category is `ORDER BY` and subjective comparisons, for instance, of multimedia and free-text descriptions. Ranking this kinds of information, e.g., identifying the best picture or assessing the informativeness of a commentary is something humans are known to excel at, at least in direct comparison with attempts to achieve the same in an automatic fashion.

[3]Recent statistics over the Linked Open Data Cloud confirm that only 10% of the resources are effectively interlinked,[4]

[5]`http://www.semantic-web.at/index.php?id=1&subid=57&action=resource&item=3217`

Finally, the actual scenario we are addressing in CrowdSPARQL is per design a good fit for microtask crowdsourcing, as we explained in more detail in [?]. The types of tasks a SPARQL query engine has to handle have a repetitive character, iterating over rows or tuples of the same kind, and processing these in a similar fashion. As such, the tasks can be broken down into smaller units (or HITs, using Mechanical Turk terminology) that can be executed in parallel by independent actors. Means to automatically control and assess the quality of the crowd-produced data can take into account existing golden standards and benchmarks, but also automatic algorithms and manually curated data [?].

3. RELATED WORK

Combining data management technology and microtask crowdsourcing has recently received some attention in the area of relational data bases [?]. Approaches such as CrowdDB [?], TurkDB [?], Deco [?], and Qurk [?] propose extensions of the SQL language, as well as new query processing techniques to master the challenges arising when relying on less deterministic computational resources such as humans in environments with clear constraints in terms of performance and accuracy. These hybrid data base systems have been instrumental in assessing the feasibility of the general idea of crowdsourced data management; they provide preliminary evidence about how specific design parameters of a microtask crowdsourcing experiment can influence costs, quality, and response time. Our work applies the lessons learned through this complementary field of research to a scenario that exhibits formally different characteristics in terms of the ways data is produced and consumed.

Query answering over Linked Data operates in an open, distributed environment; data is published in a decentralized fashion, thus inherently tending to be noisier and more heterogeneous than it is the case in traditional data bases; data sets may contain inconsistent information, especially when used in combination; their content will often overlap, hence the necessity of links between data sets identifying identical or related entities to facilitate integration. All these aspects raise different challenges for CrowdSPARQL, not necessarily in terms of the actual tasks the crowd is confronted with - these are similar to the ones the data base systems mentioned earlier tackle - but mostly with respect to the types of optimizations the resulting hybrid query engine must apply in order for the overall approach to create real added value in comparison to fully automatic query processing techniques.

This holds foremost for the number of questions which could be theoretically subject to crowdsourcing. For most real-world query answering problems over the Linked Data Cloud this number quickly reaches orders of magnitude of tens if not hundreds of HITs. Means to filter the problem space, and select only those questions which are likely to truly improve the quality of the query results at reasonable costs thus become essential. A more or less straightforward way to deal with this issue is to apply automatic algorithms that generate potential candidates for the solution to be sought by the crowd - for instance, a data interlinking algorithm identifying links between corresponding entities - but the combination between such algorithms and SPARQL query engines has not yet been investigated at full length in the Linked Data community. In fact, we expect our research to be applicable in this area as well, as we will provide insights into how specific parameters of such algorithms - most prominently the result accuracy - can play a role in the generation of query plans and in query optimization.

A second important challenge for crowdsourced Linked Data query answering is the evaluation and prediction of the quality and timeliness of answers. In order to be able to combine intermediary results produced by the crowd, by conventional SPARQL query engines, or, why not, by additional tools generating candidate solutions for specific tasks such as data interlinking or ontological classification, we need methods to analyze trade-offs between quality and time considerations.

As we progress towards a fully fledged implementation for a crowdsourcing-enabled SPARQL engine, it will be interesting to learn more about this design space in its concrete Linked-Data-specific instantiations. In addition we also plan to replicate some of the experiments and optimization heuristics proposed in crowdsourced data base research to actual data sets, as the evaluations the works cited describe is mostly based on synthetic data constructed for the purpose of the experiments. By contrast, our implementation has been tested in a real-world scenario, on data as it is made available on the Web in the Linked Open Data Cloud.

Moving towards a more technically oriented level, by comparison to the relational data base systems discussed earlier, CrowdSPARQL specifically targets graph-based representation formats and protocols, in particular Linked Data, and proposes the usage of the same technologies, extensions of SPARQL and VoID, as well as SPARQL patterns and SPIN,[6] to induce crowdsourcing functionality to Linked Data query processing. While the idea of declarative data and task descriptions has also been proposed elsewhere [**?**, **?**, **?**], the added value of CrowdSPARQL is that is does not introduce new languages or technologies, and can take into account the semantic properties of the application data and of the task descriptions, which can be inferred automatically or collected as the system is in use.

The inclusion of commonly-used predicates with well-established human-browsable representations (`rdfs:label`, `rdfs:comment`, `foaf:depiction`, `wgs84:lat`, `wsg84:long`, etc.) in such SPARQL-based task descriptions means that tasks can be given HTML-based representations simply based on existing technology.[7] This is an advantage of the use of explicit, shared semantics in the encoding of these data sets.

In addition, the solution we are proposing is equally applicable to any data management system, independently of whether it operates on Linked Data or not; the usage of semantic technologies guarantees a flexible and efficient management of such hybrid technical workflows, where human and computational intelligence offered through a multitude of services and platforms needs to be seamlessly interwoven.

A second category of related approaches focuses on a specific type of task, also relevant for Linked Data query answering. Most work in this space has concentrated on entity linking or resolutions, with systems such as ZenCrowd [**?**] and CrowdER [**?**] providing interesting optimizations along different dimensions such as quality assurance and resource management. These ideas could be easily integrated in a future version of CrowdSPARQL customized for those specific scenarios - for instance, ZenCrowd shows how probabilistic reasoning can be used in combination with paid crowdsourcing in order to improve the quality of entity extraction and linking on a corpus of news articles. Microtask crowdsourcing has shown to be feasible for several other types of scenarios related to semantic technologies. CrowdMAP [**?**] looks into how the precision and recall of existing ontology alignment algorithms could be enhanced using human labor leveraged via CrowdFlower. Their experiments give insights into the usage of this alternative microtask platform, which offers more advanced means for the assignment of workers to tasks and quality control than the Mechanical Turk. Eckert et al. have looked into the question of ontological classification via the crowd, however not in the context of SPARQL query answering. They re-construct a basic ontology, essentially a class hierarchy, in the philosophy domain using Mechanical Turk; besides insights into the means they applied to reduce spam and improve workers' productivity, their work also confirms the usefulness of such generic labor markets for tasks and domains which are further away from the common scenarios to which, say, MTurk is known to be generally applicable: looking for simple pieces of information, image labeling and alike.

Last, but not least, there is an increasing body of research available that looks into methods and techniques to improve worker productivity and HITs design, with the most promising findings being published at the annual HCOMP workshop.[8] These results are complementary to our work, as they crowdsourcing-specific optimizations rather than data management-related ones.

4. MAIN CONTRIBUTIONS

To the best of our knowledge, this paper introduces the first hybrid query engine over Linked Data that is able to execute SPARQL queries as a combination of machine and human-driven functionality. We defined an architecture for SPARQL query answering which supports microtask crowdsourcing features as a first-class computational component which aims to enhance existing Linked Data sets at query processing time, thus truly implementing the "pay-as-you-go" credo of the Linked data movement. Furthermore, our experiments provide first insights into the structure and the dynamics of such payments. The solution we are proposing is based on minimal extensions to SPARQL and VoID, and can be extended with customizations for specific domains and application scenarios. The usage of a declarative approach to data and task management enables a flexible integration of human- and machine-driven components; in addition, it considerably simplifies the generation of HTML-based HITs interfaces, and the publication of the (validated) crowd contributions as Linked Data. Last, but not least, through our experiments we provide insights into which characteristics of microtask crowdsourcing are instrumental in using this novel approach to computation in an efficient manner.

5. OUR APPROACH

CrowdSPARQL is a crowdsourcing-enabled SPARQL engine. It can (i) identify those parts of a SPARQL query, which should be subject to human-computation services; (ii) create hybrid query plans taking into account performance estimates of these services, and (iii) generate HTML interfaces of Human Intelligence Tasks that are posted to a microtask platform taking into account the most appropriate configurations in terms of payment model, workers expertise, answer redundancy, and quality assurance techniques to achieve optimal results with respect to both costs and execution time. An important aspect of any crowdsourcing experiment is quality control, as the requester of a task ideally expects to be able to identify accurate crowd answers with as little manual effort as possible. As such, CrowdSPARQL realizes a series of different methods to evaluate human contributions, identify spammers and low-effort workers, and compare experiments results with predefined gold standards and other quality benchmarks.

Figure **??** depicts the CrowdSPARQL architecture. It consists of two core components: the SPARQL query engine that automat-

[6]The idea of using SPIN to reduce the number of questions sent to the crowd based on intermediary results already received in the crowdsourcing process is out of the scope of this paper, and was briefly introduced in [**?**].

[7]See also `http://km.aifb.kit.edu/sites/spark/`.

[8]See `http://www.humancomputation.com/`

ically retrieves information from Linked Data via SPARQL endpoints, and the human-computation engine that handle the crowdsourcing parts submitted as microtasks to a crowdsourcing platform.

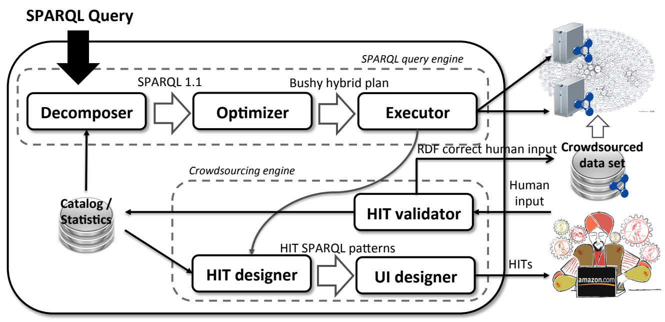

Figure 1: CrowdSPARQL architecture

When a SPARQL query is issued, the query decomposer first identifies which parts of the query will be handled through human computation, and then selects the Linked Data sets, in which the rest of the query should be evaluated. A catalog stores information about available SPARQL endpoints and VoID descriptions of data sets. We propose an extension to VoID in order to define the classes and properties of the data sets, which the Linked Data application developer expects to be commonly crowdsourced. This information, specified at application design time, is used to decompose the query. The specification of the model thus created is based on standard SPARQL 1.1, grouping the triple patterns in sub-queries using `SERVICE` blocks. The CrowdSPARQL optimizer combines the automatic and crowdsourcing sub-queries in a bushy-tree fashion to reduce the number of intermediate results, and produces a hybrid query plan. The executor evaluates the automatic sub-queries against the selected SPARQL endpoints, and invokes the human-computation engine to solve the remaining parts of the original query via microtasks. Additionally, the executor implements adaptive physical operators [**?**] to join the intermediate outcome. On the crowdsourcing side, the HIT designer creates specific types of tasks to solve the crowdsourced sub-queries, described via SPARQL patterns, and sets up the parameters to execute the microtasks taking into account the accuracy, time and cost constraints of the application. As mentioned earlier, in previous work of ours [**?**] we analyzed the architecture for applications consuming Linked Data as suggested in a recent book on the topic in [**?**] to identify candidate components that can be feasibly approached through crowdsourcing. Our system implements a subset of the corresponding tasks, namely ontological classification, entity resolution, and ordering. The remaining tasks are mainly related to labeling, translation and curation, and are planned to be integrated into a future release of the engine. For each type of task, the UI (user interface) designer produces HTML forms based on the SPARQL descriptions of the tasks, and submits the groups of HITs to the microtask platform. Input from the crowd is processed by the HIT validator, which implements quality metrics to accept or reject the assignments, and stores the curated answers in the crowdsourced data set repository as RDF. In the following sections we elaborate on the functionality of each of the core components of the CrowdSPARQL architecture.

5.1 Extensions to SPARQL and VoID

In this section, we will formally define the proposed crowdsourcing-motivated features for the SPARQL query language and the VoID vocabulary. The basic idea behind these extensions was introduced informally in previous work of ours in [**?**]. In the current implementation of our hybrid query engine, we support an extension of the `ORDER BY` operator termed `ORDER BY CROWD` to enhance the outcomes of fuzzy comparisons over Linked Data resources. As mentioned in earlier sections, the best example for such comparisons being probably pictorial representations of entities aiming to identify the *most beautiful* or the *most representative* of them, as perceived by the user.

Definition 1. **(Operator ORDER BY CROWD)** This operator consists of a new order modifier `CROWD` added after the `ORDER BY` clause. The extension to the SPARQL grammar looks as follows:

```
OrderClause ::= 'ORDER' 'BY'
(OrderCondition + | 'CROWD' OrderCondition +
SubjectiveComparison)
```

In this definition `SubjectiveComparison` is a `rdfs:Literal`, by which the user can specify the ranking question to be solved by the crowd. `OrderCondition` corresponds to the grammatical rule with the same name defined in SPARQL 1.0.

Entity resolution and ontological classification are handled through extensions of the VoID vocabulary with the class `void:CrowdSourceable` and the property `void:crowdSourceable`, by which the application developer can specify classes and properties whose instantiations are subject to crowdsourcing.[9]

Definition 2. **(Crowdsourcing a class)** In the VoID description of a data set, which is expected to be queried by our engine, a class `C` subject to crowdsourcing is defined through the following triple pattern:

```
C rdfs:subClassOf void:CrowdSourceable .
```

where `C` and `void:CrowdSourceable` are subclasses of `rdfs:Class`, and all the instances of `C` will be resolved by human contribution.

Due to the transitive property of `rdfs:subClassOf`, all the subclasses of `C` are therefore `void:CrowdSourcable`.

Definition 3. **(Crowdsourcing a property)** In the VoID description of the data set, a property `P` whose instantiations should be crowdsourced is defined through the following triple patterns:

```
P rdf:type void:crowdSourceable .
P void:crowdDomain D .
P void:crowdRange R .
```

In this definition `P` and `void:crowdSourceable` are instances of `rdf:Property`; `D`, `R`, `void:crowdDomain` and `void:crowdRange` are subclasses of `rdfs:Class`; and all the instances of `D` and `R` linked by `P` will be resolved through human input. By default, `D` and `R` are subclasses of `rdfs:Class`.

The definitions of crowdsourced classes and properties in the VoiD description are used by the query decomposer to identify the parts of the SPARQL query that are translated to HITs and posted on the microtask platform.

[9]We appreciate the potential problems arising by similarly naming the two types of primitives; however, we felt that using a terminology such as `void:crowdClass` and `void:crowdProperty`, as suggested in earlier work [**?**] would be misleading with respect to the semantics of RDFS and OWL.

5.2 SPARQL query engine

5.2.1 *Decomposer*

In this work, we extended the decomposition techniques presented in [?], which follow a two-fold approach. In the first step, the decomposer analyzes the Basic Graph Patterns (BGPs) in the `WHERE` clause of the query, and determines whether a triple pattern should be solved by human intervention based on the data set VoID descriptions. A triple pattern `{s p o .}` is subject to crowdsourcing if:

- The subject `s` or the object `o` belong to the transitive closure of the relation `rdfs:subClassOf` of the classes defined as `void:CrowdSourceable` in the VoID descriptions;
- The predicate `p` is defined as `void:crowdSourceable`, and the subject `s` or the predicate `p` are defined as `void:crowdDomain` or `void:crowdRange`, respectively.

The decomposer then heuristically identifies the available Linked Data services (SPARQL endpoints) to evaluate the remaining triple patterns, and groups them in star-shaped sub-queries that are modeled as SPARQL 1.1 `SERVICE` blocks according to the selected sources. Additionally, the crowdsourcing triple patterns are heuristically appended to the computational sub-queries with `OPTIONAL` operators allowing the binding of available data which (in some cases) can be further used as a benchmark. After all the triple patterns in the `WHERE` clause are analyzed, the decomposer then easily identifies whether the sequence modifier `ORDER BY` must be human-computed by the presence of the keyword `CROWD`.

5.2.2 *Optimizer*

The sub-queries identified in the decomposition component are combined by the optimizer to generate the execution plan. The planning techniques from [?] were extended in order to take into consideration statistics related to the performance of human-based services in the optimization phase, including monetary cost, estimated completion time, and quality. This information is collected and aggregated in every query execution, and adjusted according to the different execution scenarios. On the other hand, maintaining full up-to-date statistics about Linked Data data sets accessible via distributed and autonomous SPARQL endpoints is not always feasible [?]. Therefore our optimization techniques do not rely on information about the performance of machine-driven computational services. In order to estimate the size of the automatic result set and calculate an upper bound for the total cost of the crowdsourced sub-queries, the optimizer contacts the SPARQL endpoints on-the-fly to retrieve the cardinality of the intermediate results, and implements a greedy-based algorithm to heuristically traverse the plan space and select a sub-optimal bushy tree plan, where the number of Cartesian products and the height of the tree are minimized. The combination of decomposition and planning heuristics were emperically tested in [?] and the results suggested that they may overcome existing SPARQL engines' optimizers.

The outcome of the optimizer is a hybrid tree plan, where the leaves correspond to service blocks evaluated against automatic services (SPARQL endpoints) or human services (microtask platform), and the internal nodes are operators which combine the intermediate results. Note that the automatic query sub-plan allows the full evaluation of the original SPARQL query (excluding the `ORDER BY CROWD` operator) , i.e., every single triple pattern is included at least in a service block which is evaluated against Linked Data sets. Similarly to traditional database systems, the optimizer also selects the corresponding physical operators to efficiently evaluate the plan.

5.2.3 *Executor*

The executor evaluates the hybrid tree plan generated by the optimizer. This evaluation consists of contacting the relevant SPARQL endpoints identified in the decompositon stage to evaluate the automatic sub-plans against them, then invokes the CrowdSPARQL crowdsourcing engine in order to create the Human Intelligence Tasks (HITs) with the information retrieved from the endpoints. The intermediate results are opportunistically combined by adaptive physical operators, which are able to adjust their behavior according to the availability of the sources and produce the asnwers incrementally as data arrives. The CrowdSPARQL executor component currenty offers an adaptive version for the operators Join, OPTIONAL, UNION and PROJECT, from the SPARQL algebra. The CrowdSPARQL executor also supports the adaptive operators agjoin and adjoin [?] which are extensions of the *Symmetric Hash Join* [?] and XJoin [?], and *Dependent join* [?], respectively.

Additionally, in some cases the executor also evaluates the SPARQL patterns describing the HITs required by the UI designer in order to generate human-readable questions.

5.3 Crowdsourcing engine

5.3.1 *HIT designer*

Based on the hybrid execution plan evaluated by the executor, the HIT designer identifies the corresponding types of human tasks to submit to the crowdsourcing platform. Currently, CrowdSPARQL is enabled to handle the following types of tasks:

Entity resolution: In terms of Linked Data, this task involves the creation of `owl:sameAs` links between resources at instance level. The HIT designer is able to easily identifying the triple patterns from the crowdsourcing sub-plans which correspond to entity resolution tasks by matching this OWL predicate.

Ordering: This type of task allows the user to impose subjective ordering criteria over the Linked Data resources. The proposed SPARQL sequence modifier `ORDER BY CROWD` is considered by the HIT designer as a crowdsourcing ordering task.

Classification: This task is emphasized on the relationships between Linked Data resources, and the outcome from this task may create different types of RDF links. Classification relates to (but is not subsumed by) the entity resolution task, and the triple patterns subject to crowdsourcing whose predicates are not `owl:sameAs` are managed as taxonomical categorization.

Each crowdousricing task in the query plan is described with three arguments (IN, OUT, VARS) as follows: the required SPARQL patterns to generate human-readable questions (IN), the SPARQL patterns defining the output from the crowd (OUT), and the binding variables in each question (VARS).

The HIT designer additionally selects the appropriate parameters to efficiently evaluate each type of human task, based on statistics about rewards, granularity, completeness, latency (elapsed time to accept a task and total time to complete a sub-query) and quality from previously executed human tasks. For each type of task to be executed, this component defines the (monetary) reward, HIT lifetime and required qualifications for the crowd.

5.3.2 *UI designer*

The UI designer implements different HTML templates for each type of human task. According to the SPARQL descriptions devised by the HIT designer, the UI component instantiates the templates with the binding data as result of evaluating the SPARQL patterns specified in the INPUT argument. Usually, the variables instantiated with URIs are assigned as the values for the HTML elements within the form, while the human-redable information (labels, comments, image URL specified with the `<img>` HTML tag, etc.) is used for generating the visual part of the HITs.

The VARS argument is analyzed by the UI designer to determine the HTML elements in each question within the HITs: if the VARS set does not contains a variable between brackets, the outcome of the task is gathered through a text field in the HTML form; in the other case, the UI designer implements the question as multiple choice selection, where the options correspond to the binding data of the variable between brackets.

The HTML-based representations of the tasks are encoded as MTurk HTMLQuestion data structures and directly submitted as HIT batches or groups to the Amazon's Mechanical Turk platform.

5.3.3 HIT validator

The HIT validator is configured to periodically collect the human input from the platform, and process the retrieved results implementing spam detection techniques as well as quality metrics for each type of task. Section **??** explains in more details the CrowdSPARQL strategies for validating the outcome from the crowd.

The results determined as correct answers are transformed into RDF data according to the OUT argument defined by the HIT designer, and stored in the crowdsourced data set component.

5.4 Crowdsourced data set repository

This repository stores as RDF data the crowd answers distinguished as corrects by the HIT validator. This component may be implemented as a built-in module within the CrowdSPARQL architecture, but also it could be deployed as an external source. Based on current SPARQL capabalities, we devise three different approaches to save the HIT results:

Storage in an external triple store. In this case, a new repository is created to save the results from the crowd. To enable querying the original Linked Data set in conjunction with the derived answers from the crowd, a federated SPARQL query can be built with the `SERVICE` clause from the SPARQL 1.1 federation extension to access both sources.

Storage in a different named graph. One of the triple stores' features is the maintanance of several RDF graphs in a single repository, which allows storing different data sets or distinguishing between different versions of a single data set. Following this approach, the RDF data gathered from the crowsourced tasks can be saved in different named graph from the original data set, and both can be accesed in a single SPARQL query by specifiying the graphs in the `FROM` clause.

Storage in the same graph. This approach stores the crowd input and the Linked Data set within a single RDF graph, yet it might be no longer possible to differentiate the human-computed triples and the original ones. In order to overcome this limiation, it is necessary to annotate the resulting triples from the crowdsourced tasks.

Additionally, we propose the creation of an RDF vocabulary, to model provenance and other relevant information about the human-computed triples, i.e., creation date, number of voters, confidence in the answer, etc.