CrowdSourced Semantic Enrichment for Participatory e-Government

Francesco Di Mauro
Computer Science Dept.
Univ. of Torino
dimauro@di.unito.it

Paolo Pasteris
Computer Science Dept.
Univ. of Torino
pasteris@di.unito.it

Maria Luisa Sapino
Computer Science Dept.
Univ. of Torino
mlsapino@di.unito.it

K. Selcuk Candan
CIDSE
Arizona State University
candan@asu.edu

Giovanna Antonella Dino
Earth Sciences Dept.
Univ. of Torino
giovanna.dino@unito.it

Piergiorgio Rossetti
Earth Sciences Dept.
Univ. of Torino
piergiorgio.rossetti@unito.it

ABSTRACT
When making decisions impacting public utility and encouraging and/or enforcing behavioral rules, public administrators need to rely on data and knowledge supporting their choices, which can be used to better inform those citizens who will be affected by such decisions. Many open data repositories exist and can be accessed and used by both decision makers and citizens. Similarly, semantic tagging is now commonly used as a way to allow users provide their own knowledge to be associated to data. In this paper, we present a novel participatory system which allows traditional databases and semantic tagging modules coexist in the same knowledge base, and provides the users with query enrichment functionalities to enable ontology-based query expansion. We describe CroSSE, our CrowdSourced Semantic Enrichment system architecture, define the enrichment specification language, and discuss a use case in which the proposed technology is being applied in a participatory e-government setting. The use case is in the context of our SmartGround EU funded project, in which a relational database platform is designed to collect data of interest concerning secondary raw materials from mines as well as municipality waste. CroSSE semantic enrichment architecture interacts with this platform to expand queries and results on the basis of users’ domain knowledge.

CCS Concepts
• Information systems → Information integration; Social tagging systems;

Keywords
Crowdsourced semantic enrichment, semantic tagging and query expansion, ontologies

1. INTRODUCTION
When making decisions impacting public utility and encouraging and/or enforcing (possibly unpopular) behavioral rules, public administrators need to also rely on data and knowledge supporting their choices, which can be used to better inform those citizens who will be affected by such decisions. Often times, decision makers would need to conduct hypothetical reasoning, to estimate the implications of actions they are considering, or the impacts of new laws (such as enforcing some prohibition, or setting new thresholds in the definition of allowed activities) which they are planning to enact (“What would happen if no more than a certain amount of waste can be shipped to some specific landfill from a given region?”; “Would the available stock site still be sufficient?”; “What if some combination of elements in a landfill was considered dangerous, and its presence would trigger a fine to the manager of the landfill?”; “How many landfills would be charged high fines?”). We note that, in general, assumptions, that may vary from user to user or from location to location, may be seen as defining the context within which the database has to be queried: “Assuming that the presence of some combination of elements in a landfill might pollute the air in a certain number of kilometers in the neighbourhood, what would be the estimated polluted area, given the available data about the waste deposits?”

To allow such decision making, we are developing a participatory system which allows users to provide their own domain knowledge, and supports traditional databases and semantic tagging modules that coexist in the same platform. Moreover, we provide the users with query enrichment functionalities to enable user-provided knowledge based query expansion, thus combining existing factual data and personal knowledge. This can be exploited to promote digital interactions between institutions and citizens, facilitating their involvement in crowd sourced governance processes. In this paper, we describe CroSSE, our CrowdSourced Semantic Enrichment system architecture, define the enrichment specification language, and discuss a use case in which the proposed technology is actually applied. The use case is in the context of our SmartGround EU funded project.
in which a database platform is designed to capture data of interest concerning secondary raw materials from mines as well as municipality waste. The proposed semantic enrichment architecture, CroSSE, interacts with this database to expand queries and personalize results on the basis of users' domain knowledge. The paper is organized as follows. In Section 2 we briefly survey related literature. Section 3 introduces the motivating scenario from which the explanatory examples used along the paper will be taken, and gives context to the system architecture described in Section 4. Section 5 describes the semantics of the enriched queries, while Section 6 concludes the paper.

2. RELATED WORK

Data Integration: In general, there are three types of information-integration systems. In source-centric systems, the sources are defined in terms of the global schema and are referred to as local-as-view, or LAV, systems (Information Manifold [10], Emerac [15]). The LAV approach, while flexible, assumes a consistent integrated view. An alternative approach is to define the global schema in terms of the sources. This is called global-as-view, or GAV (HERMES [1], SIMS [2], TSIMMIS [9]), and WEBBASE [6, 7], by Davulcu). In GAV systems, whenever a source changes or is added, the global schema needs to be modified. The third class is a hybrid referred to as a GLAV system [18]. Orchestra [17] and FICSR [5] are systems that focus on managing disagreements that arise (at both schema and instance levels) during data sharing. In Orchestra, each participant has a (locally) consistent database instance, containing the set of tuples (possibly originated from other participants) that it accepts. FICSR creates a data structure that captures all interpretations of a conflicting database and can provide different views, ranked with the user's individual assumptions and preferences to different users. [19] offers a survey of different DB integration techniques. Among them, mediated query systems enable a uniform data access solution by providing a single point for read-only query of heterogeneous data sources. In this approach, the global query processor sends sub-query to local/distributed sources and manages the reconciliation of the results.

Ontology Management and CrowdSourcing: [13] and [12], focus on crowdsourcing ontology verification and engineering, in the biomedical domain. They apply ontological verification to large biomedical ontologies, in which the class hierarchy not only is the core structure, but is the only semantic relationship created by ontology developers. Using a crowdsourcing method for ontology verification (in which workers answer computer-generated questions based on ontology axioms) the hierarchy verification is subdivided in micro-tasks and the results are measured. So crowd-workers can collaborate with the domain experts, improving the quality of the enriched ontology, while reputation and altruism are forms of incentive models.

In [4] the problem of ontology-based information reuse is oriented to the realization of knowledge-based digital ecosystems. In particular, the authors present techniques based on linguistic analysis that, starting from the vocabularies contained in each source ontology and relating them with the initial (or proto) ontology, can facilitate the process of ontology construction, automating the selection and reuse of existing data models. [16] presents the NeOn methodology for ontology engineering. Without a rigid framework, this approach considers the ontological development as the construction of networks of ontologies, in which resources may be managed by people in different organizations.

Ontology Driven Query Formulation: The Ontology-Based Data Access (ODBA) is the focus of [8], devoted to the understanding of how reasoning on the ontology affects the query answering process. ODBA can be implemented as a three level architecture consisting of the ontology, the data sources, and the mapping between them. Answers to a query are not only a data structure that collects (in terms of data integration) the various sources, but also include semantically rich descriptions of the relevant concepts in the domain of interest. [14] deals with ontology-driven query formulation, in which the intensional description of a relational database is mapped to a OWL-DL description, the language in which the domain experts express their specific knowledge. On this common OWL-DL formalization, the user may formulate ontological queries that are then translated into the corresponding relational SQL statements.

Available ontologies can be used in web site management and integration scenarios; in particular, [11] describes a Semantic portAL (SEAL) which presents a three-layer architecture encompassing: (a) heterogeneous data sources (DB, XML, HTML); (b) a wrapper that aggregates the sources in a common data model; (c) integration modules (and specific mediators for the dynamic case) able to reconcile the data sources. The ontology can offer support to user query targeted to different sources, and the intensive use of schema information can facilitate the activity of integration, selection and presentation requested by a web tool that is based on a semantic conceptual model. The central aspect of this family of semantic portals (and other similar system, like SmartGround) is the help offered to a community of users, each one contributing to the global knowledge base while also consuming the common enriched knowledge.

3. MOTIVATING APPLICATION

Within the context of the SmartGround European project, we are developing a databank platform in which a broad spectrum of data relevant to decision making in the context of waste management are collected. It is widely recognized that large amounts of waste that - if properly recovered from industrial, mining and municipal landfills - could provide economic gains are instead regularly lost because of poor waste management practices. SmartGround aims to enable best practices in the context of reuse of waste materials, through an innovative data collection strategy that makes easily accessible information about available waste materials and their potential reuse opportunities.

The SmartGround databank aims at integrating the already existing information from the national and international databases (national agencies, public bodies data bases, European statistics). The information stored in the platform might be seen as describing different real word scenarios, each of them described by a consistent interpretation of the database. Due to the geographic distribution of such data sources, the different times at which data have been collected, and different data recording standards and strategies, existing data sources are often heterogeneous, incomplete and incomparable.

Decision makers will leverage the collected information

1 http://www.smart-ground.eu/index.php
and reason about the implications of the reality the collected data represent. They might also perform hypothetical reasoning, possibly within different contexts representing, for example, the rules and constraints enforced in different countries. Moreover some users may enrich the information stored in the database with their own knowledge, to personalise queries and reasoning tasks. For example the director of a specific laboratory might be interested in combining the information about the analysis on a landfill (information stored in the database), with other information relevant to her but not stored in the database (for example, the data and role in the laboratory of the person who has signed the analysis report), and might be interested in querying the database in the context of her personal knowledge.

To support such user participation, we are developing a semantic tagging module (see Section 4) in which users can insert their own knowledge (and possibly share knowledge acquired from other users’ statements) knowledge enriches and extends the information already available in the database and the conclusions that can be drawn from it.

Figure 2: RDF based knowledge statement insertion at CroSSE

3.1 The Case for Semantic Enrichment

Figure 1 briefly sketches the structure of CroSSE platform. We distinguish between (1) data, which are stored in the database and represent factual information shared by the different partner institutions and taken as certain knowledge by all the users, and (2) personal, contextual knowledge, which reflects the users’ interpretation of the data, or the available meta-knowledge that the users might want to use in combination with the stored data. While data are commonly accepted as true, personal knowledge reflects the users’ individual experience and know-how. The semantic tagging interface provides users the option of inserting their own knowledge (in the form of RDF statements, see Figure 2) or accepting as their own (part of) the contextual knowledge already inserted by other users. The inserted knowledge enriches and extends the information already available in the database and the conclusions that can be drawn from it.

Example 3.1. The schema of our SmartGround database includes tables to collect information about the elements, minerals and/or chemical compounds that can be found in different mine landfills. While those waste items are described in terms of their chemical properties and of the available amounts in the various considered landfills, the database schema does not capture, for example, information about the different labs which conducted the analyses, and their internal hierarchical organizations. Similarly, information about the fact that some elements (maybe if co-located with some others) might be considered as pollutant might depend on local (to the states or the regions) rules and regulations, fixing thresholds for acceptable amounts of specific elements in space units. The semantic tagging module will allow interested users to extend the knowledge base by annotating data with such information. Once suitably tagged, the users can query the database and obtain information about the existence of pollutant elements (extracted from the database) in some areas, according to the analysis conducted by some specific lab (information derived from the semantic knowledge base). We refer to section 5 for the presentation of the query formulation and semantics.

Notice that there is no centralized control on the correctness and/or consistency of the crowdsourced knowledge. This is because we aim to give each user the freedom to express her own beliefs, assumptions, or hypotheses about the domain, and query the database within the context of such additional information not readily available in the database.

3.2 The SmartGround Use Case

In order to illustrate the capabilities of CroSSE semantic tagging module, we will refer in the following sections to an extremely simplified (and provisional) version of the SmartGround database, which represents a sample data fragment capturing the knowledge of what is contained (in basic terms of elements, minerals and chemical compounds) in four mine landfills. Although simple, this database provided a useful scenario to simulate example queries.
An ontology (expressed as RDF statements), formalizing the common knowledge available in the system along with what each user may define as her knowledge, according to her domain of expertise, enriches the factual information in the database. For example, Figure 4 provides classification of the possible types of waste that can be found in a mine landfill. Some of these concepts can be a direct reference to the database content (Mineral, Element, ChemicalCompound), while others can be part of the general common knowledge (Metal) or can be user defined concepts (identified in this example by a dashed frame, as HazardousWaste). The ontology concepts are connected by means of RDF properties which can be predefined (e.g. isA, representing a parent-children relationship) or introduced (and potentially shared) by the users of the system (identified by dashed lines in this example, as oreAssemblage).

In the example scenario, the oreAssemblage property (ideally defined by a user who has a geology background knowledge) associates to a given element/mineral the set of what other elements/minerals are usually found in the type of rocks (and hence in the mining waste) from which the element/mineral is extracted (e.g. Cobalt and Copper can usually be found along Nickel, see Figure 4). This information can thus be exploited to infer the content of a landfill even when it is not directly specified in the database.

Figure 5 provides another example where the enriched knowledgebase defines the relationship between the technicians working in a certain lab and the landfills in the database, where the signsReportFor is a user contributed property, defined to keep track of the information on which technician has signed the analysis report for a given landfill.

Figure 6 illustrates the part of the ontology that keeps track of which concepts, properties, and statements have been defined by which user. As can be seen in this example, elements can be shared among the users, such as the concept HazardousWaste and the property oreAssemblage being shared by two users. Whenever a user interacts with the system, only the elements of the ontology which are common, shared with or defined by her will be accounted in the semantic enrichment process. It is important to notice that the semantics associated to the elements created by the users is unknown to the system.

### 4. SYSTEM ARCHITECTURE

In this section, we introduce the architecture of CroSSE (as reported in Figure 7).

- The commonly shared factual information is stored in a relational database, $DB$. Figure 3 shows a fragment of the SmartGround relational database, in which data about the different landfills and the elements, minerals and chemical compounds contained in them are stored.

- The knowledge base, $KB$, contains a set of RDF statements. RDF statements may or may not be shared. Each statement is annotated with information about its “source”, the users who inserted it into the system and the users who have chosen to accept this statement as theirs. Given the set of all possible users, $U$, the knowledgebase can be logically seen as a mapping which associates each user to the set of RDF statements she believes in (either by contributing the statement directly, or by accepting other users statements): $KB : U \rightarrow P(KB)$, $KB(u) = \{s = (subj, pred, obj) \mid u$ believes in $s\}$. 

---

**Figure 4: Sample fragment of supporting (partly user provided) knowledge base**

**Figure 5: Landfill and laboratories ontology**

**Figure 6: Keeping track of users and their contributions to the knowledgebase**

**Figure 7: The architecture of CroSSE**
For each user, u, the knowledge she will rely on while querying the system will be $DB \cup KB(u)$.

With a slight abuse of notation, in this paper we refer to $KB$ both as the mapping which associates each user to the statements she believes in and as the global set of available RDF statements, with the intended meaning that $KB = \bigcup_{u \in U} KB(u)$

- The two information stores, $DB$ and $KB$, need to interact, so that each user $u$ can perceive their combination as a single, integrated, data source. The semantic query module, SQM has the following role:
  - it submits SPARQL queries to $KB$, possibly with input parameters resulting from an SQL query to the relational $DB$;
  - it submits SQL queries to $DB$, possibly with input parameters resulting from a SPARQL query to $KB$;
- Wrappers provide interface functionalities between the SQM module and the two information stores, possibly generating local internal views (see Section 5) which are used to produce the final answer to the user’s query.

5. QUERIES SEMANTIC ENRICHMENT

In this section, we describe the semantic enrichment process, and present examples to highlight how enriched queries can be used to combine the information stored in a database with ontological domain knowledge, exploiting both the common knowledge and the user provided notions.

5.1 Query Enrichment Process

The basic idea behind query enrichment consists in exploiting the ontology in order to identify semantic patterns not directly recognizable in the knowledge contained in the database, thus providing the user a more informative result set, which contains data derived by the common/shared knowledge along with the one that the user herself put in the system. The RDF formalism allows, through the SPARQL language, to navigate the ontology by following the edges of the graph, thus allowing the definition of complex paths connecting various concepts. In particular, the user can choose one ore more attributes (from a relation involved in the query or from the query result set) and run a SPARQL query for any of its values, to obtain a set of replacements (which may or may not contain the initial value according to the user preferences) to enrich the scope of the query or the result set itself. The process involves the following steps:

- Step 1: The user specifies one of the two supported enrichment modalities: (1) In the first modality, the user formulates an SQL query to the factual database $DB$, and asks the system to enrich the query by extending or replacing the values of a subset of the output attributes with corresponding values resulting from a SPARQL query to $KB$. In this first case the enrichment only plays a role to introduce new/replace existing values in the query output; (2) In the second enrichment modality, through a SPARQL statement, the user specifies concepts/values that she believes should appear as values for some database attributes (although this may not be the case in $DB$). The user intention is to have, as the result of her SQL query, the very same results she would have if the specified enrichment values were indeed stored in the specified database attributes. In this case, the extension/value replacement is intended to happen also during the “internal” phases of the query processing, for example when evaluating join operations between different relations.
- Step 2: The extension/replacement requirements are specified. This is obtained by defining a mapping between relational attributes/attribute values from $DB$ and concepts/concept instances from $KB$.
- Step 3: The wrapper creates the temporary views which are needed to evaluate the specified extended query, in the required modality.
- Step 4: The query is evaluated, results are returned and the temporary extension views are released.

In the following subsections, we provide details of the above steps, and we describe how the user can enrich the result of a given SQL query by extracting the information contained in the ontology through one or more SPARQL queries.

5.2 Query Enrichment Semantics

In this subsection we formalize the semantics of the query enrichment process.

5.2.1 Replacement Set

Let $Q_{SQL}$ be an SQL query on the database $DB$ and $A_j$ in relation $R_i$ be an attribute referred to in the query. For semantic enrichment, we allow the user to associate a replacement set for the attribute $R_i.A_j$

$$RS(R_i.A_j) = \{ \langle a_1, r_{1,1} \rangle, \langle a_1, r_{1,2} \rangle, \ldots, \langle a_m, r_{m,1} \rangle, \ldots, \langle a_m, r_{m,n_m} \rangle \}$$

which specifies, for each constant $a_k \in dom(A_j)$, the set of values $\{ r_{k,1}, \ldots, r_{k,n_k} \}$ to replace $a_k$. The replacement set is obtained through a SPARQL query $Q_{SPARQL}$ interrogating the semantic knowledge base, $KB$, for replacements for values in the domain of $R_i.A_j$. The replacement set for the values of $R_i.A_j$ are passed back to the $DB$ in the form of a replacement table $RT_{i,j}(V, V')$

Here the attribute $V$ contains the initial values and $V'$ the corresponding replacements. One important aspect to consider is that there is no guarantee that the values returned by the SPARQL query belong to the same domain of the original value, and this is particularly true for the RDF properties which connect concepts of different classes. While this does not represent a critical problem in the enriched SQL query evaluation (these values will simply not match with any value in the involved relation), the user is expected to pay particular attention both in the ontology enrichment and in the SPARQL query definition in order to guarantee overall consistency.

Note that an SQL query to $DB$ consists of different clauses. The WHERE clause specifies the conditions necessary to be satisfied for returning a tuple, whereas the SELECT clause specifies the set of attributes to be returned. Enrichment can be applied to both of these clauses as discussed next.

5.2.2 Enrichment by SELECT Clause

This form of semantic enrichment directly operates on the results of a given SQL query, where the operations are performed upon completion of the query evaluation process.
Let $Q$ be a query with $A_1, \ldots, A_r$ attributes specified in the SELECT clause. Let $RES(A_1, \ldots, A_r)$ be the result table containing a set of $n$ result tuples:

$$RES(A_1, \ldots, A_r) = \{ (a_1^1, \ldots, a_r^1), \ldots, (a_1^n, \ldots, a_r^n) \}$$

Let us also be given a replacement table $RT(V, V')$ (obtained through a SPARQL query $Q_{SPARQL}$ to the database) for attribute $A_i$. Given this, we can have two distinct query enrichment strategies: (a) In schema extension, the result has a new attribute combining information from the DB with information from the KB:

$$RES'((A_1, \ldots, A_r, V', \ldots, A_r) = RES \Join_{A_i = v'} RT$$

(b) in attribute replacement, however, the original values specified in $A_i$ are replaced with the corresponding values specified in the replacement table:

$$RES'(A_1, \ldots, A_{i-1}, V', A_{i+1}, \ldots, A_r) = \Pi_{A_1, \ldots, A_{i-1}, V', A_{i+1}, \ldots, A_r} (RES \Join_{A_i = v'} RT)$$

It is to be noticed that the first strategy makes sense only if a valid value for the additional attribute $V'$ can be assigned to each tuple in the result set.

**Example 5.1 (Schema Extension).** Let us consider the SmartGround scenario reported in Section 3.2 and let us suppose that the user is initially interested in the content of the landfill ‘c’ in terms of elements, interrogating the system through an SQL query $Q_{SQL}$ in the form:

```sql
SELECT Lname, Ename
FROM Landfill, EleCont
WHERE Landfill.Lname = EleCont.Lname AND Landfill.Lname = 'c'
```

The result of the query is reported in Table 1(a).

Let us further assume that the user is interested in enriching the query result by specifying for each returned element whether it is potentially hazardous for the environment or not. In order to do so, a new attribute Hazard is added in the result set and the value of this result attribute is collected through SPARQL queries to the knowledge base.

$$RT (Ename, Hazard) =
SELECT ?Ename ?Hazard
WHERE { ?Ename isA Element .
BIND(EXISTS { ?Ename isA HazardousWaste
as ?Hazard } )
}$$

The result of the enriched query is reported in Table 1(b).

**Example 5.2 (Attribute Replacement).** Let us assume that the user is interested in the waste content (in terms of both elements and minerals) of the landfill ‘b’:

<table>
<thead>
<tr>
<th>Lname</th>
<th>Ename</th>
<th>Hazard</th>
</tr>
</thead>
<tbody>
<tr>
<td>c</td>
<td>mercury</td>
<td>yes</td>
</tr>
<tr>
<td>c</td>
<td>cobalt</td>
<td>no</td>
</tr>
</tbody>
</table>

(a) Original

<table>
<thead>
<tr>
<th>Lname</th>
<th>Elements</th>
<th>Hazard</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>nickel</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>zinc</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>gold</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>fluorsite</td>
<td></td>
</tr>
<tr>
<td>b</td>
<td>barite</td>
<td></td>
</tr>
</tbody>
</table>

(b) Enriched

The result of this query is reported in Table 2(a). Now let us further assume that the user wants to know what other elements (or minerals) could be found in the same landfill considering the knowledge on how they usually come combined in the types of rocks from which they are extracted during mining operations (a knowledge expressed in the ontology by the property oreAssemblage, as explained in section 3.2). This corresponds to the SPARQL query that collects for each element or mineral connected to waste results through the oreAssemblage property:

$$RT (Waste1, Waste2) =
SELECT ?waste1 ?waste2
WHERE { ?waste1 oreAssemblage ?waste2
}$$

The result of the enriched query is reported in Table 2(b). Since, in Figure 4 the Zinc element is connected to both Fluorite and Barite minerals, the result set is enriched with two additional tuples, as reported in Table 2(b). ♦

### 5.2.3 Enrichment by WHERE Clause

As we discussed earlier, enrichment process may also be applied to WHERE clause, during the condition evaluation. In this case, depending on whether they appear in positive or negative conditions, the replaced values contribute in the query processing by extending or restricting the domains of the involved variable(s): Intuitively,

- positive conditions (such as equality) are to be considered as satisfied (i.e. true) whenever they are satisfied by at least one of the replacement values, while
- negative conditions (such as non-equality) are to be considered as satisfied (i.e. true) whenever they are not satisfied by any of the replacement values.

**Replacement of Constants.**

Let us consider an SQL query posed to the database, $DB$, of the form:

```sql
SELECT ...
FROM ...
WHERE \( \theta(A, 'c') \) ...
```
which contains a condition involving an attribute $A$ and a constant 'c'. Let $RT$ be the replacement table for this attribute. Depending on the nature of the predicate $\theta$, the condition will be rewritten as follows:

- $A = 'c'$: This condition will be rewritten as
  \[
  A \text{ IN (SELECT } V' \text{ FROM } RT \text{ WHERE } V = 'c')
  \]

- $A \neq 'c'$: This condition will be rewritten as
  \[
  A \text{ NOT IN (SELECT } V' \text{ FROM } RT \text{ WHERE } V = 'c')
  \]

- $A < 'c'$, $A \leq 'c'$: These conditions will be rewritten as
  \[
  A < \text{ MAX (SELECT } V' \text{ FROM } RT \text{ WHERE } V = 'c')
  \]

- $A > 'c'$, $A \geq 'c'$: These conditions will be rewritten as
  \[
  A > \text{ MIN (SELECT } V' \text{ FROM } RT \text{ WHERE } V = 'c')
  \]

We next provide an example.

**Example 5.3 (Constant Replacement).** Suppose that, after extracting the waste content of landfill ‘b’ (Example 5.2), the user is now interested in the elements and the minerals contained in all the landfills whose analysis reports were signed by technicians who work for the same laboratory as the one who signed the report for ‘b’.

As illustrated in Figure 5, this information is formalized in the ontology by means of the properties signsReportFor and employedIn. The following SPARQL query identifies a path on the ontology graph which, starting from ‘b’ navigates the hierarchy defined for the laboratories personnel to identify the landfills which meet the specified criteria:

$$RT \ (\text{Lname}, \ \text{Lname2}) =$$

$$\text{SELECT} \ \text{landfill} \ ?\text{landfill1} \ \text{WHERE} \ \{ \ ?\text{landfill} = \ 'b' \ \text{filesReportFor} \ ?\text{landfill} \ . \ ?\text{tech1} \ \text{employedIn} \ ?\text{lab} \ . \ ?\text{tech2} \ \text{employedIn} \ ?\text{lab} \ . \ ?\text{tech2} \ \text{filesReportFor} \ ?\text{landfill1} \}$$

The replacement table $RT$ (Table 3(a)), containing the results of the SPARQL query, is then exploited for the query enrichment:

\[
\text{SELECT} \ \text{Landfill.Lname, EleCont.Lname AS waste FROM Landfill, EleCont WHERE Landfill.Lname = EleCont.Lname AND Landfill.Lname IN (SELECT Lname2 FROM RT) UNION SELECT Landfill.Lname, MinCont.Mname AS waste FROM Landfill, MinCont WHERE Landfill.Lname = MinCont.Mname AND landfill.Lname IN (SELECT Lname2 FROM RT)}
\]

Table 3(b) contains the result of this enriched query.

**Replacement of Variables.**

Consider an SQL query to the database, $DB$, of the form

\[
\text{SELECT ... FROM ... WHERE ... } \theta(A, B) ... \]

which contains a condition involving attributes $A$ and $B$. Let $RT_B$ be the replacement table for attribute $B$. Depending on the nature of the predicate $\theta$, the condition will be rewritten as follows:

- $A = B$: This condition will be rewritten as
  \[
  A \text{ IN (SELECT } V' \text{ FROM } RT_B \text{ WHERE } V = B)
  \]

- $A \neq B$: This condition will be rewritten as
  \[
  A \text{ NOT IN (SELECT } V' \text{ FROM } RT_B \text{ WHERE } V = B)
  \]

- $A < B$, $A \leq B$: These conditions will be rewritten as
  \[
  A < \text{ MAX (SELECT } V' \text{ FROM } RT_B \text{ WHERE } V = B)
  \]

- $A > B$, $A \geq B$: These conditions will be rewritten as
  \[
  A > \text{ MIN (SELECT } V' \text{ FROM } RT_B \text{ WHERE } V = B)
  \]

Let us next consider the case where both attributes involved in the condition evaluation are provided with replacement tables, $RT_A$ and $RT_B$. In this case, the condition will be written as follows:

- $A = B$: This condition will be rewritten as
  \[
  \text{IS NOT EMPTY( (SELECT } V' \text{ FROM } RT_A \text{ WHERE } V = A) \ \text{INTERSECT (SELECT } V' \text{ FROM } RT_B \text{ WHERE } V = B) )}
  \]

- $A \neq B$: This condition will be rewritten as
  \[
  \text{IS EMPTY( (SELECT } V' \text{ FROM } RT_A \text{ WHERE } V = A) \ \text{INTERSECT (SELECT } V' \text{ FROM } RT_B \text{ WHERE } V = B) )}
  \]

- $A < B$: This condition will be rewritten as
  \[
  \text{MIN (SELECT } V' \text{ FROM } RT_A \text{ WHERE } V = A) \ < \text{ MAX (SELECT } V' \text{ FROM } RT_B \text{ WHERE } V = B) \]
  \[A \leq B \text{ is also rewritten similarly.}\]

- $A > B$: This condition will be rewritten as
  \[
  \text{MAX (SELECT } V' \text{ FROM } RT_A \text{ WHERE } V = A) \ > \text{ MIN (SELECT } V' \text{ FROM } RT_B \text{ WHERE } V = B) \]
  \[A \geq B \text{ is also rewritten similarly.}\]

We next provide an example.

<table>
<thead>
<tr>
<th>Lname</th>
<th>Lname2</th>
<th>waste</th>
</tr>
</thead>
<tbody>
<tr>
<td>b</td>
<td>b</td>
<td>gold</td>
</tr>
<tr>
<td>b</td>
<td>c</td>
<td>nickel</td>
</tr>
<tr>
<td>b</td>
<td>d</td>
<td>zinc</td>
</tr>
<tr>
<td>c</td>
<td>cobalt</td>
<td></td>
</tr>
<tr>
<td>c</td>
<td>mercury</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>iron</td>
<td></td>
</tr>
<tr>
<td>d</td>
<td>zinc</td>
<td></td>
</tr>
</tbody>
</table>
Table 4: Result for the SQL query of Example 5.4
<table>
<thead>
<tr>
<th>Lname1</th>
<th>Lname2</th>
<th>Ename</th>
</tr>
</thead>
<tbody>
<tr>
<td>a</td>
<td>d</td>
<td>iron</td>
</tr>
<tr>
<td>d</td>
<td>a</td>
<td>iron</td>
</tr>
<tr>
<td>b</td>
<td>d</td>
<td>zinc</td>
</tr>
<tr>
<td>d</td>
<td>b</td>
<td>zinc</td>
</tr>
</tbody>
</table>

Table 5: $RT$ table for Example5.4
<table>
<thead>
<tr>
<th>Element</th>
<th>Element1</th>
</tr>
</thead>
<tbody>
<tr>
<td>nickel</td>
<td>nickel</td>
</tr>
<tr>
<td>nickel</td>
<td>cobalt</td>
</tr>
<tr>
<td>zinc</td>
<td>zinc</td>
</tr>
<tr>
<td>zinc</td>
<td>fluorite</td>
</tr>
</tbody>
</table>

Example 5.4 (VARIABLE REPLACEMENT). As an example, let us consider the case in which the user is interested in selecting from the database the landfills that contain the same elements. The following SQL query to the DB returns the required information, as shown in Table 4.

\[
\begin{align*}
\text{SELECT } & \text{EleCont1.Lname AS Lname1, EleCont2.Lname AS Lname2, EleCont1.Ename} \\
\text{FROM } & \text{EleCont1, EleCont2} \\
\text{WHERE } & \text{EleCont1.Ename = EleCont2.Ename} \\
\text{AND } & \text{EleCont1.Lname \neq EleCont2.Lname}
\end{align*}
\]

Let us further assume that the user is interested in extending the result set by also including those elements that, although not listed in the DB as being contained in the considered landfills, are declared as possibly contained in them in the knowledgebase, as stated by the oreAssemblage property in the crowdsourced knowledgebase:

\[
\text{RT(Element, Element1) =}
\begin{align*}
\text{SELECT } & \text{?element ?element1} \\
\text{WHERE } & \text{?element oreAssemblage ?element1 }
\end{align*}
\]

RT(Element, Element1), is shown in Table 5.

The SQL query satisfying the extension request is the following.

\[
\begin{align*}
\text{SELECT } & \text{EleCont1.Lname AS Lname1, EleCont2.Lname AS Lname2, EleCont1.Ename} \\
\text{FROM } & \text{EleCont1, EleCont2} \\
\text{WHERE } & \text{EleCont1.Ename IN (SELECT RT.Element FROM RT) AND EleCont1.Lname \neq EleCont2.landfill}
\end{align*}
\]

Table 6 presents the extended results, including the information extracted directly from the database, and the information extracted leveraging the ontological knowledge.

6. CONCLUSIONS

In this paper, we presented CroSSE, our Crowd Sourced Semantic Enrichment query system architecture, defined the enrichment specification language, and discussed a use case in the context of our SmartGround EU funded project. We focused on a declarative presentation of CroSSE’s functionalities. As future work we are planning to focus on operational optimization oriented issues, including the study of how the materialization of the replacement table RT can be avoided, for optimization purposes.

7. REFERENCES