REPRESENTATION AND QUERIES

ANDRIANA GKANIATSOU

1. Translation of the Incoming Query

The incoming query, which is described in predicate logic, needs to be translated into an appropriate format to query the datasets. The datasets are in XML/RDF and relational databases.

1.1. **XML/RDF.** The SEPA datasets consist of XML/RDF files. There are two approaches for quering this format: treat it as XML or, treat it as the union of XML and RDF. Traditionally, XQuery, XPath are used to query XML. In [1] XSPARQL, a new language which is considered as the union of SPARQL and XQuery, is proposed. However, following an XML-based query approach entails more challenges. Based on [2], since XML languages are Turing complete, mapping Turing Complete languages to First Order Logic and *vice versa* is already a hard problem. (There are many proposals on this mapping, but I am not sure whether we want to go that deep).

Another solution, is to consider the SEPA datasets as pure RDF and convert it to n3 format (which is equivalent to RDF). Such a conversion can be achieved (automatically) by rdfcat. By doing so, we can query the datasets using SPARQL. In [3] is shown that SPARQL is as expressive as Relational Algebra. In that way it is easier to tranlsate the incoming predicate into SPARQL than XQuery, or another xml-based query language.

1.2. **Relational Databases.** Since relational databases are based on first-order predicate logic, translating the incoming query into an SQL (or another database query language) query should be straightforward.

1.3. **Mapping between the Languages.** When translating a relational database to predicates, each row of a table is a predicate and the predicate's name is the table's name. The values of the columns of the table are the arguments of that predicate. Similarly, we map a SPARQL query to a predicate. However, the translation from one format to the other is not always easy. This is better shown through an example. Consider the following incoming query, described in predicate logic:

```
surfaceWaterBodies("West Highland", WaterBodyName, SepawId )
```

When we translate is in SQL, we end up with the following query:

```
SELECT ?, ??
FROM surfaceWaterBodies
WHERE{
???  = "West Highland"}
```

instead of[1]:

```
SELECT waterBodyName, sepawId
FROM surfaceWaterBodies
WHERE{
subBasinDistrict = "West Highland"}
```

We face the same problem when we tranlsate the query in SPARQ. The resulting query is:

```
SELECT ?x ?y
FROM <surfaceWaterBodies.rdf>
WHERE {
?x ??  "West Highland" ;
?x ???  ?y}.
```

instead of[2]:

```
SELECT ?x ?y
FROM <surfaceWaterBodies.rdf>
WHERE {
?x sepaw:subBasinDistrict "West Highland" ;
sepaw:waterBodyName ?y}.
```

In a SPARQL query, a variable's name does not affect the SELECT statement, not the WHERE statement. Also, WHERE statement can also have blank nodes. However, in SQL and in any other database query language SELECT and WHERE statement cannot have variables or blank nodes. To avoid this problem we need to make some assumptions about the type of the data we are searching for (columns for a SQL query/ predicates for a SPARQL query). We can either assume that the variables represent also the type, or that we receive additional data about the types.

2. Translation of the Datasets

In this Section we will consider translating the datasets or some parts of the datasets, into first-order syntax. The cases that we consider are: (i) translate the whole dataset, (ii) translate subset(s) of the datasets, (iii) no translation at all.

2.1. **Translate Whole Dataset.**

This approach is the most beneficial regarding the diagnosis process. However, the datasets can be huge, so this approach could be very time consuming and highly costly.

2.2. **Translate Subsets of the Datasets.**

Our aim is to translate the smallest possible subset that contains the appropriate information for the diagnosis and the repair process. We identify two cases: (i) the schema of the dataset, (iii) a subset that contains related with the query information.

Dataset Schema The schema of the dataset is the minimal dataset that provides the appropriate information for the diagnosis and repair. However, obtaining the schema of the dataset means that we have access permissions to do so. Consider Table 2, and the incoming query:

[1]Example is taken from the SEPA dataset

[2]Example is taken from the SEPA dataset

NHS Board	Site Type	Location Name	Location Address	File Type	Comments

TABLE 1. Schema of NHS Table

NHS Board	Site Type	Location Name	Location Address	File Type	Comments
Highland	ED	Belford Hospital	BELFORD ROAD, FORT WILLIAM, PH33 6BS	E	-

TABLE 2. NHS Table: Data describes the available NHS Emergency Departments.

```
hospital(highland, HospitalName, HospitalAddress)
```

Under the assumption that we have access rights, we obtain the schema shown in Table 1. To translate the schema into a first-order predicate, we use the table's name as the predicate name. The labels (column names) we obtain from the schema represent the classes of the arguments of this predicate. That means, that we cannot translate the labels as arguments (constants) of the predicate. Instead, we add variables and we denote the classes that these variables belong to. So, after the translation process we should have something like:
`nhs(NHSBoard, SiteType, LocationAddress, FileType, Comments)`, `instance(NHSBoard, nHSBoard) instance( SiteType, siteType), instance(LocationName, locationName), instance(LocationAddress, locationAddress), instance(FileType, fileType), instance(Comments, comments)` The types of the variables can indicate possible synonyms, hypernyms *e.t.c.*

Query Related Subset We can identify such subset based on the synonyms, hypernyms *e.t.c.* of the keywords the query contains. Another way is to create a subset with all the dataset entries that contain at least one of the query's keywords. This approach is more appropriate when the mismatch occurs at the values of the datasets, rather than the labels.

2.3. **Case 1. No translation.** None of the datasets is translated; all remain in their original format (relational databases and rdf triples). Consider Table 2, described in as a relational database, and the following incoming query (in first-order):
`hospital(highland, HospitalName, HospitalAddress)`
This query will be translated into an SQL query[3]

```
SELECT hospitalName, hospitalAddress
FROM hospital
WHERE {hospitalArea = "Highland"}
```

[3]Under the assumption that we have all the information that we need about the types. Note that this information might not be the correct one.

The above query, will not return any results because: (i) The table name is incorrect, (ii) The column names are incorrect. However, SQL server will only provide an ERROR message *e.g. Table does not exist* and no other feedback.

Secondly, even if we had the appropriate feedback, based on that query language, the only kind of mismatches we could identify would be semantic mismatches. But even in this case, will we be able to identify the appropriate repairs? For instance, following the previous example, we do not have any information on whether the synonym we are looking for is domain specific or not.

3. System Overview

To handle the previous requirements, until now, we have concluded to two programming languages PROLOG and JAVA. With PROLOG we can process and translate[4] the incoming query. Also PROLOG is appropriate for the diagnosis and the repair of the mismatch. JAVA can be used to handle the SQL and SPARQL queries. The outcome of the queries can then be sent back to the PROLOG system for further process. An overview of the system is presented in Image 1. **Questions**

We consider only first-order predicates as incoming queries. So, for each incoming predicate we need to: (i) choose the format that we will translate it, (ii) decide the annotation of the query, *i.e.* how many synonyms to include, the domain *etc.*, (iii) decide whether the annotation will take place before or after the translation process, before or after the query is sent (before or after failure).

In case the original query fails, how do we determine how many synonyms we would like to find? Also, assume that we have n synonyms: are we going to send all n queries? only some? or we will keep on sending until something returns an answer?

Another challenge, regarding the synonyms, is how do we determine the domain that we will look at? Suppose that we consider three specific domains. Searching for synonyms in all domains is not time-efficient. Can we assume that the user provides the domain?

References

[1] W. Akhtar, J. Kopecký, T. Krennwallner, and A. Polleres, "Xsparql: traveling between the xml and rdf worlds - and avoiding the xslt pilgrimage," in *Proceedings of the 5th European semantic web conference on The semantic web: research and applications*, ESWC'08, pp. 432–447, 2008.

[2] W. Fan, "Querying and Storing XML, University Of Edinburgh," 2012.

[3] R. Angles and C. Gutierrez, "The expressive power of sparql," in *Proceedings of the 7th International Conference on The Semantic Web*, ISWC '08, pp. 114–129, 2008.

[4]JAVA is also appropriate for implementing the translation process.

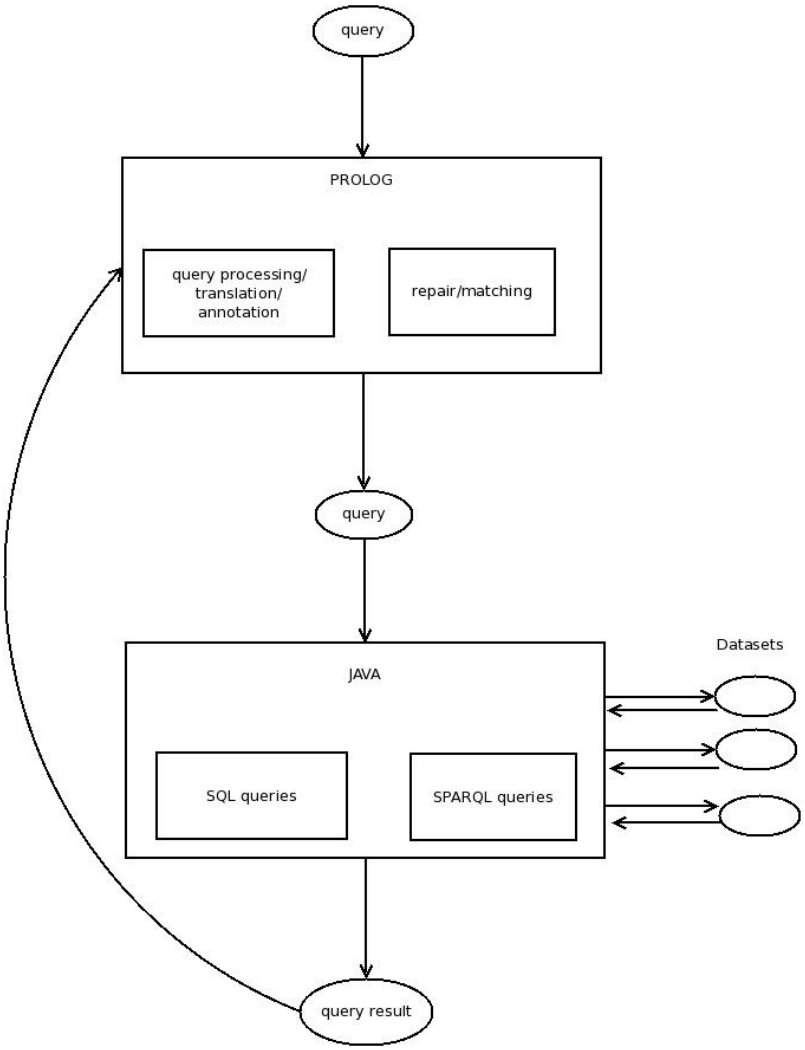

FIGURE 1. System Overview