OpenHIE Entity Matching Service

OHIE Architecture Quick Links

Architecture Subcommunity Call

Architecture Governance and Principles

Architecture Review Board Deliverables, Members, and Responsibilities

Architecture Road Map

OpenHIE Entity Matching Service

Specification Management and Workflow Processes

OpenHIE Standards and Profiles

Purpose

The purpose of the entity matching service is to enable matching in a single list of **patients**, **health workers**, **facilities** or other entities or to find potential matches between two lists of the same entities.

How it Works

The service receives a FHIR message with the entity to be matched and returns zero to 10 matches and their scores. We are supporting FHIR through Hapi FHIR.

The service can also be executed as a standalone program using flat data files. It can search for duplicates within a single file, or it can search for matches between two files.

Potential Use Cases

We envision the following potential use cases:

1. Ensuring the the entity doesn't exist when entering a new instance of the entity
2. Duplicate checking during bulk imports.
3. Analysis of potential duplicates in an existing data set.
4. Mapping one data set of entities to their corresponding value in another data set.

Potential Implementations

Depending upon the use case, we envision that there might be a spectrum of implementation options. We expect to learn from the first implementations and refine the use patterns based upon experience. For now, we imagine the following types of architectural implementations:

1. Tight coupling - A tightly coupled implementation might be one where the matching service software library is incorporated into the architecture component.
2. Medium - This type of implementation could be one where the service interacts directly with the architecture component's data source.
3. Loose - This type of service may load data into the service's data base and analyze the data from there.

FHIR Reference

http://gforge.hl7.org/gf/project/fhir/tracker/?action=TrackerItemEdit&tracker_item_id=9685&start=0

High Level Overview of Mapping Service Components

Sample URL:

https://testmap.ohie.org/registry/fhir/Location/$match

Sample Request:

```
<Parameters xmlns="http://hl7.org/fhir"> <parameter> <name value="location"
/> <resource> <Location xmlns="http://hl7.org/fhir"> <contained> <Location
xmlns="http://hl7.org/fhir"> <id value="1"/> <identifier> <value value="a.
bc.1.sample"/> </identifier> <name value="simple health"/> </Location> <
/contained> <identifier> <value value="117"/> </identifier> <name value="
simple clinic"/> <position> <longitude value="10"/> <latitude value="100"
/> </position> <partOf> <reference value="#1"/> </partOf> </Location> <
/resource> </parameter> <parameter> <name value="count"/> <valueInteger
value="5"/> </parameter> </Parameters>
```

Sample Response:

```
<Bundle xmlns="http://hl7.org/fhir"> <entry> <resource> <Location xmlns="
http://hl7.org/fhir"> <id value="1000010"/> <contained> <Location xmlns="
http://hl7.org/fhir"> <id value="con31"/> <identifier> <value value="A.BC.
1.SAMPLE"/> </identifier> <name value="SAMPLE HEALTH"/> </Location> <
/contained> <extension url="http://ohie.org/fhir/StructureDefinition/datim-
mechid"> <valueString value="1111"/> </extension> <identifier> <value
value="117"/> </identifier> <name value="SIMPLE CLINIC"/> <position>
<longitude value="10.0"/> <latitude value="100.0"/> </position> <partOf>
<reference value="#con31"/> </partOf> </Location> </resource> <search>
<score value="0.997621798717855834404133474890841171145439147949218 75"/> <
/search> </entry> </Bundle>
```


Matching Engine Source Code:

https://tools.regenstrief.org/stash/users/amartin/repos/registry/browse

Matching Approachs

There are multiple ways to determine a match.

1. Score - you can set the algorithm to provide a score. This approach can use thresholds. This method is currently implemented in the service.
2. States (match, non-match, manual review) - this approach can use rules to establish rules for matching. Some of the base capabilities in in place and Regenstrief is working to mature this feature.
3. Score and States - Some services provide for mixing these states.

Potential Workflow - Find Possible Matches as an HIE Service

Example Actors:

- Entity Authority -
  - OpenInfoMan/InterLinked Registry with the FHIR adapter
- Entity Searcher
  - iHRIS - when a new health worker record is added
  - DHIS2 - when a new facility is added
  - OpenMRS - when a new client is added

This is one example of a possible workflow:

Unknown macro: 'uml-sequence'

Interfaces

Different interfaces will need to be created to instantiate different use cases that call the service.

Matching Algorithms

While the entity matching service currently implements a sophisticated probabilistic algorithm, a key overarching goal of the entity matching service is to accommodate a variety matching methods. The current algorithm can be configured for matching different types of entities.

Configuration File

The matching service is highly configurable.

Page - Quick Links

1. Model configuration:
   One will need to configure the service to understand the fields and data types that exist in the database.
   Mappings to FHIR fields can also be configured.
2. Matching configuration:
   One can define matching rules using boolean logic for deterministic matching, or one can assign weights for probabilistic matching.
   Implementers can run RecMatch or another service to compute the configuration weights.

Importer

When configuring the matching service to run against an existing database, one will likely have existing tools for loading data into the database. However, an importer is included with the matching service. This can be helpful if one creates a new database to be used by the matching service. The importer can take a flat file and import data into the database.

Data Structure

The matching service can run against a single flat table. It can also run against hierarchical structures. For example, one might have a table named patient. If a patient can have multiple identifier numbers from different domains, then there might be a separate child table named patient_identifier, where each row contains the identifier value itself, the value's domain, and a reference to a patient row. One can configure the matching service to understand both tables and the relationship between them. Then values from both tables can be used for patient matching.

Example Workflow

Input - 2 source files to be compared (or a file/Database with duplicates)

The engine is capable of processing different file types (csv, tsv etc).We need two files we want to match in the similar column structure.

```
Facility Name,Region,District,Council,Ward,Latitude,Longitude,Facility
Type,Pepfar/MOH
abcFacilityName, abcRegion, abcDistrict, abcCouncil, abcWard, abcLatitude,
abcLongitude, abcFacilityType, M/P
```

Configuration File

We need to create a configuration file where we depict the column structure, mention which algorithm to use against each column we want to consider for comparison, mention if we want a score shown for potential matches, mention above what score the matches can be shown, also if we want to ignore values for comparison instead of considering them and penalizing the score.

Sample configuration

```
<registry>
   <properties>
    <property>
      <key>org.regenstrief.registry.score.MeanMetric.nullSkipped</key>
      <value>true</value>
    </property>
  </properties>
<tables>
<table>
  <name>FACILITY</name>
  <candidateMetric>
    mean(
      lcs("DISTRICT"),
      lcs("FACILITY_NAME"),
      lcs("REGION"),
      lcs("COUNCIL"),
      lcs("WARD"),
      euclidean("LATITUDE", "LONGITUDE", 0.1)
    )
  </candidateMetric>
```

```
  <matchEvaluator>candidate(0.3)</matchEvaluator>
  <columns>
     <column>
      <index>0</index>
      <name>FACILITY_NAME</name>
      <type>Varchar</type>
      <size>100</size>
    </column>
    <column>
      <index>1</index>
      <name>REGION</name>
      <type>Varchar</type>
      <size>100</size>
    </column>
    <column>
      <index>2</index>
      <name>DISTRICT</name>
      <type>Varchar</type>
      <size>100</size>
    </column>
    <column>
      <index>3</index>
      <name>COUNCIL</name>
      <type>Varchar</type>
      <size>100</size>
    </column>
    <column>
      <index>4</index>
      <name>WARD</name>
      <type>Varchar</type>
      <size>100</size>
    </column>
    <column>
      <index>5</index>
      <name>LATITUDE</name>
      <type>Real</type>
    </column>
    <column>
      <index>6</index>
      <name>LONGITUDE</name>
      <type>Real</type>
    </column>
    <column>
      <index>7</index>
      <name>Facility type</name>
      <type>Varchar</type>
      <size>100</size>
    </column>
    <column>
      <index>8</index>
      <name>Pepfar/MOH</name>
      <type>Varchar</type>
      <size>100</size>
    </column>
  </columns>
</table>
</tables>
</registry>
```

Run Configuration

program variables

```
-file path/to/file1 -file2 /path/to/file2 -blocking.mode blockingModeToUse
-skip.header booleanValue -candidate.max
MaximumNumberOfPotentialValuesToBeShown -include.score booleanValue -delim
fileDelimiter(ifUsingAFileInput) -table TableType
```

VM Arguments

```
-Dorg.regenstrief.registry.configuration=path/to/configuration/file
```

Output File

This file is generated at the same location as the source files with an suffix of Match. The file structure is similar to the source file but, potential matches for a specific row are displayed below the row indented.

```
abcFacilityName, abcRegion, abcDistrict, abcCouncil, abcWard, abcLatitude,
abcLongitude, abcFacilityType, M
    abcFacilityName, abcRegion, abcDistrict,abcCouncil, abcWard,
xyzLatitude,  xyzLongitude, xyzFacilityType, P, 0.8400000000000001
```

Questions and Answers

Q: Is blocking used?

A: The matching service is divided into two basic steps: coarse blocking and fine-grained matching handled in Java. The blocking step is for performance, so that the service doesn't need to apply the fine-grained matching algorithm to every row in the database. It's less flexible than the fine-grained matching step and is designed to allow fast queries based on typical database indexes. For example, an index on the name column will make this query fast:

select * from organisationunit where name=?

But a normal database won't be able to quickly run a query to search for rows based on a Levenshtein score. The <blockingScheme> element defines how the matching service will handle this coarse blocking.

Q: How is case matching supported?

The <caseMode> element can be used with these possible values:

- CASE_SENSITIVE - it allows values to be stored as mixed case, and it uses query parameters however they're received.
- QUERY_UPPER - it allows work with mixed case values in the database. It will also work regardless of the case of incoming query parameters. When doing lookups, it will convert the database values and the query parameters to upper case within the query itself, so queries will be case insensitive: select … from organisationunit where upper(name)=upper(?) But if you have a normal index on that column instead of a function-based index, then it won't be able to use the index.
- QUERY_LOWER
- STORE_UPPER - expects values to be stored as upper case in the database, and it converts query parameters to upper case before doing lookups.
- STORE_LOWER