Culture Brokers

Instance Matching Tool

Description

Information Systems Laboratory (ISL)

Institute of Computer Science

Foundation for Research and Technology - Hellas (FORTH)

Heraklion Crete

8 September 2014

Author: Elias Tzortzakakis ([tzortzak@ics.forth.gr](mailto:tzortzak@ics.forth.gr))

Contents

[Introduction 3](#_Toc397972278)

[Query Sequences preface 4](#_Toc397972279)

[Sequence steps and sequence similarity by example 5](#_Toc397972280)

[User Configuration xml file 8](#_Toc397972281)

[Results Threshold 8](#_Toc397972282)

[Literal Minimum Similarity Value 8](#_Toc397972283)

[Target Source Choice 9](#_Toc397972284)

[Source Input Files / Target Input Files 10](#_Toc397972285)

[Query Sequences Configuration 11](#_Toc397972286)

[Program output 12](#_Toc397972287)

# Introduction

Instance Matching tool is a java api developed in java 7 used in order to retrieve similarities of instance pairs, on user specified fields among “**Source Data”** and **“Target Data”**. Source Data are always a set of rdf files while Target Data may be another set of rdf files or an online Database. Currently the only online databases supported are the British Museum Collection Database [[link]](http://collection.britishmuseum.org/sparql) and CLAROS database [[link]](http://data.clarosnet.org/sparql/).

The entry point of Instance Matching Api tool is its base java class IMAPIClass and is instantiated using 3 path values:

1. The base folder path against which relative paths of source or target rdf files will be computed.
2. The full path of a User Configuration xml file. In this document we will refer to this file as UserConfiguration.xml.
3. The full path of the xml file used to declare query prototypes. In this document we will refer to this file as QueryPrototypesConfiguration.xml.

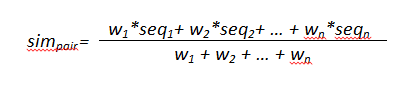
**UserConfiguration.xml**: used from any user that wishes to use the tool in order to define things like similarity threshold, input and target files, comparison mode, instances of interest, and query sequences over these instances. These options will be explained later in this document.

**QueryPrototypesConfiguration.xml**: this file should only be changed by administrator users as it defines the query prototypes that may be parameterized in UserConfiguration.xml, the known cidoc crm compatible namespaces and some cidoc crm extensions that may be used on queries performed over source and target data. These query prototypes will be transformed to valid SPARQL queries that will be performed over the Source and the Target data. Each query prototype uses a special syntax that is tested to lead in valid queries, thus it should not be changed by any user that wishes to use the api. Therefore no further information will be provided in this document. Each time a new query prototype is needed users should communicate with an administrator user that will provide them with the new query prototype configuration file that matches their request.

One basic configuration that user must do is to select the class of instances that he is interested in finding their similarity. This is accomplished by providing the cidoc class identifier e.g E39 for [E39\_Actor](http://www.cidoc-crm.org/cidoc-crm/E39_Actor) or E21 for [E21\_Person](http://www.cidoc-crm.org/cidoc-crm/E21_Person). Then the rest of the user configuration will be used in order to find source – target instance pairs that appear to have similarity above a user specified threshold.

## Query Sequences preface

User specifies the fields he is interest in by defining a set of weighted (wi) query sequences for which a sequence similarity value (seqi) will be calculated. The overall similarity (simpair) of a source – target instance pair with different uris is calculated using the following formula:



As for a source – target instance pair with the same uri, it always has an overall similarity of 1.0.

User defined Query sequences are parameterized by the following data:

1. The mnemonic string that specifies which query prototype will be used.

E.g. “uri(a) -> literal”

*Note that this mnemonic string is not analyzed by the api it is just used as an id to specify the query prototype that will be used*

1. A weight value greater than or equal to 0.0 and less than or equal to 1.0 that determines the affection that this query sequence will have to the overall similarity result of each pair. E.g. 0.5

*Note that if weight is set to 0.0 then this query sequence will be ignored and the underlying queries will not be performed at all.*

1. A set of predicate id values that will replace each prototype parameter in the selected query prototype. E.g. P131

Each parameterized query sequence is analyzed in one or more query steps that will be performed in order to collect the relevant data for each instance that will later be used in order to calculate the sequence similarity seqi of Source and Target data. In general query steps can be identified in the mnemonic string by their separating “->“ character sequence when not included in parentheses. So in the example above we can understand that mnemonic string “uri(a) -> literal” defines two query steps i.e. step A 🡪 “uri(a)” and step B 🡪 “literal”.

## Sequence steps and sequence similarity by example

The declaration of user defined query sequence that will be used as an example to further explain the way it works will be the following. We will also assume that user has specified that he is interested in [E21\_Person](http://www.cidoc-crm.org/cidoc-crm/E21_Person) instances.

<querysequence weight="**0.5**" mnemonic="**uri(a) -> literal**">

<parameter name="**a**">P131</parameter>

</querysequence>

The parameterized query sequence that occurs from this declaration is:

**uri(P131) -> literal** which defines the following two query steps:

**Step A 🡪 uri(P131):**

At this point a query will be performed in order to collect person instances along with uris which are connected to person instances through cidoc property P131. A namespace search will occur in order to retrieve the exact name of P131 property which is P131\_is\_identified\_by or P131i\_identifies and then a SPARQL query like the following (not exactly that) will be formulated and executed.

SELECT ?personUri ?P131connectedUri

{

?personUri a crm:E21\_Person .

?personUri P131\_is\_identified\_by ?P131connectedUri .

}

Thus appelation uris related by P131 to person instances will be collected for source data whereas for target data these uris will only be collected iff there is a matching uri found in the source data.

**Step B 🡪 literal:**

At this point another query will be performed which is a logical sequence of the previous in step A in order to collect person instances along with literals that are connected to the previous step uris. Thus a SPARQL query like the following (not exactly that) will be formulated and executed.

SELECT ?personUri ?P131literal

{

?personUri a crm:E21\_Person .

?personUri P131\_is\_identified\_by ?P131connectedUri .

?P131connectedUri rdfs:label ?P131literal .

}

Finally similarity for two source-target person instance pairs with different uris is calculated for this sequence by performing comparison of all results that occurred in the query steps defined in the sequence. Depending on the value type collected in each step a different comparison is performed.

Each step of each query sequence may lead to three data types:

* **uri:**

Between a uri value valA of a source person instance and another uri value valB of a target person instance, a comparison is performed to see if these values are equal or not resulting in a similarity value of 1 or 0.

* **literal:**

Between a literal value valA of a source person instance and another literal value valB of a target person instance, a comparison is performed that includes the following string matching algorithms:

* + Char Frequency Similarity
  + Digram Similarity
  + Trigram Similarity
  + Soundex Similarity
  + Edit Distance Similarity
  + Single Error Similarity

The similarity of valA and valB for this source-target pair of persons is set to be the average result of these algorithms.

* **timespan:**

Between a timespan value valA of a source person instance and another timespan value valB of a target person instance, a comparison is performed in order to see if valB overlaps or is equal to valA, thus resulting in a similarity value of 1 or 0. Note that a timespan consists of a start date and an end date each following the format yyyy-mm-dd. In cases where only one of the two is defined then the other is set to the same value as the first.

In order to find the sequence similarity (seqi) of a source-target person instance pair, all similarities of the values found in every step of the sequence are computed and then the maximum is selected to be the seqi value of the pair. Thus if a uri similarity of 1.0 is found in step A then the sequence similarity of the pair is set to seqi regardless of the values found (if any) in step B.

A more complex query step might be defined as below

<querysequence weight="**0.4**" mnemonic="**uri(a1 | a1 -> a2) -> uri(b)**">

<parameter name="**a1**">**P14**</parameter>

<parameter name="**a2**">**P9**</parameter>

<parameter name="**b**">**P108**</parameter>

</querysequence>

The above results in the parameterized query sequence

uri(P14 | P14 -> P9) -> uri(P108) which results in the following two steps

**Step A 🡪 uri(P14 | P14 -> P9):**

This query returns for each person all the events that were carried out by him, or the events that are included in an event that was carried out by him. A SPARQL equivalent would be:

SELECT ?personUri ?stepAproductionUri

{

?personUri a crm:E21\_Person .

{

?personUri crm:P14i\_performed ?stepAproductionUri.

}

UNION

{

?personUri crm:P14i\_performed ?intermediateProductionUri .

?intermediateProductionUri crm:P9\_consists\_of ?stepAproductionUri .

}

}

**Step B 🡪 uri(P108):**

This query returns for each person all the objects uris that are connected to the previous step A uri by cidoc Property P108 🡪 P108\_has\_produced. A SPARQL equivalent would be:

SELECT ?personUri ?stepBproductUri

{

?personUri a crm:E21\_Person .

{

?personUri crm:P14i\_performed ?stepAproductionUri.

}

UNION

{

?personUri crm:P14i\_performed ?intermediateProductionUri .

?intermediateProductionUri crm:P9\_consists\_of ?stepAproductionUri .

}

?stepAproductionUri crm:P108\_has\_produced ?stepBproductUri .

}

# User Configuration xml file

This is the file that user must configure in order to use the Instance Matching Api Tool. It includes the following:

## Results Threshold

<ResultsThreshold>**0.7**</ResultsThreshold>

Here the user must provide a decimal value greater than 0.0 and less than or equal to 1.0 determining the threshold that will be used in order to qualify source-target instance matching pairs.

In this example pairs that present overall similarity greater than or equal to 0.7 will qualify to the final result set.

## Literal Minimum Similarity Value

<literalMinimumSimilarityValue>**0.4**</literalMinimumSimilarityValue>

Here the user must provide a decimal value greater than 0.0 and less than or equal to 1.0. This value will be used when performing literal comparison so that literal pairs with average similarity less than this value are ignored.

The purpose of this parameter is three-fold:

1. Users may only be interested in exact matching of literals which they can accomplish by setting this value to 1.0.
2. Due to the algorithms used in String matching comparisons, the results that qualify may differ from what is expected by user. Adjusting this value gives the user the opportunity to get the desired result.
3. It may result in significant memory optimization especially when target data come from an online database. Literals returned are at once compared for similarity to any value of the same sequence and step found in all source files and if they do not qualify they are ignored. Otherwise they are kept in memory until all query sequences and all their steps are processed, in order to calculate the final similarity of each source target pair.

Suggested value: 0.4

## Target Source Choice

<TargetSourceChoice>BRITISH\_MUSEUM\_COLLECTION</TargetSourceChoice>

Here the user selects the target data that will be compared to the source rdf data files. Currently there are three options provided:

* FILE\_COMPARISON:

With this choice target data is set to be the set of rdf files that is declared in TargetInputFiles xml field.

* BRITISH\_MUSEUM\_COLLECTION:

With this choice target data is set to be the online British Museum

collection SPARQL endpoint:

<http://collection.britishmuseum.org/sparql>

* CLAROS:

With this choice target data is set to be the CLAROS SPARQL endpoint:

<http://data.clarosnet.org/sparql/>

*WARNING!!! CLAROS database does not support inference i.e. subClasses and subProperties are not included in the result set thus predicates used should be exactly these that are used in the database.*

*i.e. If asking for* [*E39\_Actor*](http://www.cidoc-crm.org/cidoc-crm/E39_Actor) *this database will not also qualify the* [*E21\_Person*](http://www.cidoc-crm.org/cidoc-crm/E21_Person) *instances as it should since E21 is subClassOf E39.*

*Also, if asking for* [*P1*](http://www.cidoc-crm.org/cidoc-crm/P1_is_identified_by) *predicate this database will not qualify the* [*P131*](http://www.cidoc-crm.org/cidoc-crm/P131_is_identified_by) *predicates as it should since P131 is subPropertyOf P1.*

## Source Input Files / Target Input Files

[<SourceInputFiles>](file:///C:\Users\tzortzak\Desktop\ImDeliverable\UserConfiguration.xml)

<File PredicateDirection=”direct”>C://…fullPath…/testdata/cidoc1.rdf</File>

….

<File>C://…fullPath…/testdata/cidoc4.rdf</File>

</SourceInputFiles>

[<TargetInputFiles>](file:///C:\Users\tzortzak\Desktop\ImDeliverable\UserConfiguration.xml)

<File>C://…fullPath…/testdata/cidoc6 - Copy.rdf</File>

</TargetInputFiles>

These two elements contain the relative path to the files that will be used as the source and the target respectively. Files declared inside TargetInputFiles elements will only be used if TargetSourceChoice is set to FILE\_COMPARISON. Relative paths use as base path the one declared in the initialization of base api class – the IMAPIClass.

It is recommended that all files contain the minimum set of namespaces needed because the instance matching tool will try to retrieve their schema and find the exact name of the predicates that will be used in the queries.

Each file element may also include a "PredicateDirection" attribute that may have one of the following values:

* "direct"
* "inverse"
* "both" (default value if none declared)

and determines the direction of the predicates that will be used.

i.e. if a query contains predicate with id P1 then:

* in case of "direct": "P1\_is\_identified\_by" will be used
* in case of "inverse": "P1i\_identifies" will be used
* in case of "both": "P1\_is\_identified\_by" and "P1i\_identifies" will be used

If inference is included in the cidoc compatible namespace of the file then "direct" choice may be used safely otherwise "both" option should be chosen or the PredicateDirection attribute not set at all.

e.g. The <http://erlangen-crm.org/current/> cidoc compatible namespace leads to an owl file that contains statements about inverse predicates thus enabling inference in queries. So files using this namespace may just use the "direct" PredicateDirection option.

The <http://www.cidoc-crm.org/cidoc-crm/> does not contain the inverse property statements so the "both" PredicateDirection option must be used unless user is certain that only one of the two directions is used throughout the file. If "inverse" option is selected then inverse predicates will be used wherever possible meaning that for P3 predicate which does not have an inverse Property the P3\_has\_note will be used.

## Query Sequences Configuration

<QuerySequencesConfiguration BaseClassID="**E21**">

<querysequence weight="**0.4**"

mnemonic="**uri(a) -> uri(b) -> timespan(c1 - c2)**">

  <parameter name="**a**">P98</parameter>

  <parameter name="**b**">P4</parameter>

  <parameter name="**c1**">P82a</parameter>

  <parameter name="**c2**">P82b</parameter>

  </querysequence>

… More querysequence elements …

</QuerySequencesConfiguration>

**BaseClassID Attribute:**

Here user declares the cidoc identifier that will be used to select the class of instances that he is interested in finding their similarity. This is accomplished by providing the cidoc class identifier e.g E39 for [E39\_Actor](http://www.cidoc-crm.org/cidoc-crm/E39_Actor) or E21 for [E21\_Person](http://www.cidoc-crm.org/cidoc-crm/E21_Person).

**Querysequence Elements:**

Here the user may specify multiple query sequences that will specify the fields of that he is interested in finding similarities. User defined Query sequences are parameterized by the following data:

1. The mnemonic string that specifies which query prototype will be used.

E.g. “uri(a) -> literal”

*Note that this mnemonic string is not analyzed by the api it is just used as an id to specify the query prototype that will be used*

1. A weight value greater than or equal to 0.0 and less than or equal to 1.0 that determines the affection that this query sequence will have to the overall similarity result of each pair. E.g. 0.5

*Note that if weight is set to 0.0 then this query sequence will be ignored and the underlying queries will not be performed at all.*

1. A set of predicate id values that will replace each prototype parameter in the selected query prototype. E.g. P131

**Parameter Elements:**

Here the user specifies the predicate ids that will replace the step variables (a, a1 .., b,b1,… c, c1… etc) of each query sequence. Apart from the standard cidoc predicate ids user can also select one of the following values which are considered as very common cidoc extensions:

* P81a resulting in P81a\_end\_of\_the\_begin
* P81b resulting in P81b\_begin\_of\_the\_end
* P82a resulting in P82a\_begin\_of\_the\_begin
* P82b resulting in P82b\_end\_of\_the\_end

Furthermore, user can use any other predicate id that is or will later be defined in the the cidoc compatible namespace declared in source or target rdf files, or the <http://erlangen-crm.org/current/> namespace that is used in the British Museum collection database. It just has to be defined in both source and target data namespaces.

# Program output

The tool performs the following steps:

1. Displays the stating date time and the selections that user has provided through the user configuration xml file.
2. If comparison to an online database is selected it attempts to specify if the database is available.
3. It collects all the namespaces from all rdf files that will be used in order to retrieve each namespace’s model.
4. Starts processing of each file which includes the transformation of the query prototype to a valid SPARQL query for each file. This operation includes the time consuming retrieval of the exact predicate names meaning that for P1 P1\_isidentified\_by and P1i\_identifies must be retrieved.
5. After processing of all input files has finished it displays the data that were found in all source files as a result of all query sequences and steps.
6. Then if comparison to the database is selected it prepares the queries for the database as in step 4 for files.
7. Finally presents the following column list where
   1. Pair similarity
   2. Source file
   3. Source instance uri
   4. Target File or Target Database File
   5. Target Instance uri

Each row of this result set is followed by the similarities found in each sequence and the values that returned that similarity.