CHAIn Documentation

Table of Contents

[Objectives of CHAIN 2](#_Toc487046609)

[Running CHAIN 3](#_Toc487046610)

[File Structure 4](#_Toc487046611)

[MATCH\_STRUC.JAVA 6](#_Toc487046612)

[BASIC FUNCTIONALITY 6](#_Toc487046613)

[PRIVATE VARIABLES/FIELDS 6](#_Toc487046614)

[METHODS 7](#_Toc487046615)

[ASSOCIATED TESTING DOCUMENTS 9](#_Toc487046616)

[CALL\_SPSM.JAVA 10](#_Toc487046617)

[BASIC FUNCTIONALITY 10](#_Toc487046618)

[PRIVATE VARIABLES/FIELDS 10](#_Toc487046619)

[METHODS 10](#_Toc487046620)

[ASSOCIATED TESTING DOCUMENTS 11](#_Toc487046621)

[BEST\_MATCH\_RESULTS.JAVA 12](#_Toc487046622)

[BASIC FUNCTIONALITY 12](#_Toc487046623)

[PRIVATE VARIABLES/FIELDS 12](#_Toc487046624)

[METHODS 12](#_Toc487046625)

[ASSOCIATED TESTING DOCUMENTS 13](#_Toc487046626)

[NODE.JAVA 14](#_Toc487046627)

[BASIC FUNCTIONALITY 14](#_Toc487046628)

[PRIVATE VARIABLES/FIELDS 14](#_Toc487046629)

[METHODS 14](#_Toc487046630)

[ASSOCIATED TESTING DOCUMENTS 14](#_Toc487046631)

[CREATE\_QUERIES.JAVA 15](#_Toc487046632)

[BASIC FUNCTIONALITY 15](#_Toc487046633)

[PRIVATE VARIABLES/FIELDS 15](#_Toc487046634)

[METHODS 15](#_Toc487046635)

[ASSOCIATED TESTING DOCUMENTS 15](#_Toc487046636)

# Objectives of CHAIN

The CHAIn (Combining Heterogeneous Agencies’ Information) system dynamically re-writes queries to databases when mismatches led to query failure.

In general, queries to databases only succeed if they are correctly writing according to the schema of that database, which is only possible if the schema of the database is known in advance. During an emergency response (and, indeed, many other situations), this is plausible in some situations, but emergencies are characterised by their unpredictability and such foresight is not always possible. If one wishes to broadcast a query to all relevant agencies, or to a new agency, or to an previous collaborator who has updated their data sources, for example, a query may fail even if there is relevant data in the data source.

CHAIn sits locally within an agency. When a query to the data sources of that agency fails, CHAIn will use various matching techniques to determine on-the-fly whether there is mismatch between the schema of the query and the schema of the database and, if so, whether there is anything in the database that matches, or approximately matches the query. If so, CHAIn rewrites the query according to the schema of the database and sends back an appropriate response (or responses), together with a score indicating how good the match is and information about the assumptions and approximations made in the match.

# Running CHAIN

1. After downloading the CHAIn project, run the following four files that can be found in *spsm/s-match-source* called *0cloneSMatch, 2buildSMatch, 3createBinary, 4extractBinary.*

Doing this should create the following folders in that directory*: s-match-parent, s-match-core, s-match-io, s-match-logic, s-match-nlp, s-match-nlp-annotation, s-match-nlp-opennlp, s-match-spsm, s-match-wordnet, s-match-examples, s-match-examples, s-match-utils, s-match-spsm-prolog.*

1. Open Eclipse and select *File > New > Project > Java Project* and enter the name for the project and uncheck the *Use Default Location* radio button and set the location to point to where this project was downloaded to. You will now be able to access and run any source and testing files.

# File Structure

TBC

# TESTING RESULT DOCUMENTS

All tests will write their results to a text file that can be found in the *outputs/testing* folder. These files have been structured to give details about each individual test including what schemas CHAIn is being run with, what the expected result is and what the actual result is that is returned.

Please note that there are some tests that will return empty results either because there have been no matches found between the source and target schema. Some tests will return the message, *Empty Results,* due to SPSM not returning anything to the application after being called.

description of implementation files

# MATCH\_STRUC.JAVA

## BASIC FUNCTIONALITY

Data structure that has been created to store the results that are returned after calling SPSM. The structure allows for easy interaction to find out information about the match details with a certain target schema.

## PRIVATE VARIABLES/FIELDS

***similarity***

This is the variable that stores the similarity value between the source and this current target schema. (double)

***dataset***

This variable holds the name of the target schema that this structure is representing the results for. (String)

***matches***

List of arrays containing the match concepts between the source and this current target schema. (ArrayList of String arrays)

***repairedSchemaTree***

Tree structure that represents the repaired schema in a tree form. (Node)

***repairedSchema***

This variable holds the repaired schema in a format that can easily be read. (String)

***numMatches***

Stores the total number matches which is calculated by the number of elements in the matches ArrayList. (int)

## METHODS

***public Match\_Struc (double sim, String targetSchema)***

Constructor that is responsible for initialising the Match\_Struc variable with the similarity value and the name of the target schema.

***public void setSimilarity (double sim)***

Sets the similarity of this match to be the value passed in as the parameter, *sim.*

***public void setDatasetSchema (String targetSchema)***

Sets the target schema to be the String that is passed in as the parameter, *targetSchema*.

***public void addMatch (String[] match)***

Responsible for adding another match concept to the list of matches stored in the ArrayList<String[]> called *matches.*

***public int getNumMatches ()***

Returns the total number of matches based on the number of elements inside *matches*.

***public double getSimValue ()***

Returns the similarity value for that target schema.

***public String getDatasetSchema ()***

Returns the name of the target schema.

***public ArrayList<String[]> getMatches ()***

Returns the ArrayList of arrays containing the match concept details.

***public String[] getMatchAtIndex (int index)***

Returns the specific match concept detail at the specified index passed as a parameter, *index*.

***public void setRepairedSchemaTree (Node schemaTree)***

Sets the repaired schema tree from the parameter, *schemaTree*.

***public void setRepairedSchema (String stringSchema)***

Sets the String version of the repaired schema from the parameter, *stringSchema*.

***public String getRepairedSchema ()***

Returns the String version of the repaired schema.

***public Node getRepairedSchemaTree ()***

Returns the repaired schema as a tree structure.

## ASSOCIATED TESTING DOCUMENTS

*SPSM\_Test\_Cases.java*

*Match\_Results\_Test\_Cases.java*

*SPSM\_Filter\_Results\_Test\_Cases.java*

*Task1\_Test\_Suite.java*

# CALL\_SPSM.JAVA

## BASIC FUNCTIONALITY

This class is responsible for initially taking in both the target and source schemas before calling SPSM with these schemas. It will then store the results as an ArrayList of Match\_Struc objects.

## PRIVATE VARIABLES/FIELDS

***targetList***

Used temporarily for splitting the target schema String if there are more than one target schema within the String. This allows the application to then call SPSM for each individual target schema. (*String[]*)

## METHODS

***public ArrayList<Match\_Struc> getSchemas (ArrayList<Match\_Struc> results, String sourceSchema, String targetSchema)***

Retrieve the schemas from the user, either through the console or as a parameter which gets written to their respective files. From there we can start to make a call to SPSM with the current target.

***public ArrayList<Match\_Struc> makeCallToSPSM (ArrayList<Match\_Struc results, String currTarget)***

Makes the call to SPSM using a bash file called *call-spsm.sh.* This should allow for SPSM to run and then write the results out as a serialised object to *serialised-results.ser*. It will then call the appropriate method to read this file and store the results in a Match\_Struc object.

***public ArrayList<Match\_Struc> recordSerialisedResults (ArrayList<Match\_Struc> results, String targetSchema)***

Responsible for reading the serialised object from *serialised-results.ser* and calls the appropriate method that will then go through the different pieces of information stored within this object and stores what is required for CHAIn.

***public ArrayList<Match\_Struc> readObject (ArrayList<Match\_Struc> results, String targetSchema, IContextMapping<INode> mapping)***

Reads the information stored in the object read in and picks out the data that is required for use by CHAIn.

***public String getNodePathString (INode node)***

Responsible for taking in a *INode* object and turns it into something readable so that we can store the match concepts in a convenient way.

## ASSOCIATED TESTING DOCUMENTS

*SPSM\_Test\_Cases.java*

*SPSM\_Filter\_Results\_Test\_Cases.java*

*Task1\_Test\_Suite.java*

# BEST\_MATCH\_RESULTS.JAVA

## BASIC FUNCTIONALITY

Responsible for taking results after calling SPSM and then filtering to ensure each result is over the determined threshold, cut the number of results to the desired *n* and sort them so that the match with the highest similarity is at the top of the list.

## PRIVATE VARIABLES/FIELDS

-

## METHODS

***public ArrayList<Match\_Struc> getThresholdAndFilter (ArrayList<Match\_Struc> results, Double threshVal, int limNum)***

Start off by taking in the results, threshold value and the number of results wanted but if these values haven’t been passed in through parameters, then ask the user to enter them through the console. It will then strip off any matches that are lower than the threshold value that has been entered.

***public ArrayList<Match\_Struc> sortResultingMatches (ArrayList<Match\_Struc> filteredRes, int limitNo)***

This method will sort the resulting matches after they have been filtered to ensure that only those greater than or equal to the threshold value are left. These matches are sorted so that the matches with the highest similarity value is at the top of the list.

***public void displayResults (ArrayList<Match\_Struc> res)***

This method is used to print out the results after they have been filtered and sorted.

## ASSOCIATED TESTING DOCUMENTS

*Match\_Results\_Test\_Cases.java*

*SPSM\_Filter\_Results\_Test\_Cases.java*

*Task1\_Test\_Suite.java*